Physics-based surface energy model optimization for water bodies in cold climates using visible and calibrated thermal infrared imagery

by

May V. Casterline

B.S. Rochester Institute of Technology, 2006
A.S. Rochester Institute of Technology, 2006

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Chester F. Carlson Center for Imaging Science
College of Science
Rochester Institute of Technology

November 22, 2013

Signature of the Author

Accepted by

Coordinator, Ph.D. Degree Program Date
The Ph.D. Degree Dissertation of May V. Casterline has been examined and approved by the dissertation committee as satisfactory for the dissertation required for the Ph.D. degree in Imaging Science.

Dr. Carl Salvaggio, Dissertation Advisor

Dr. Alfred Garrett

Dr. Edward Hensel

Dr. John Schott

Date

II
Title of Dissertation:

Physics-based surface energy model optimization for water bodies in cold climates using visible and calibrated thermal infrared imagery

I, May V. Casterline, hereby grant permission to Wallace Memorial Library of R.I.T. to reproduce my thesis in whole or in part. Any reproduction will not be for commercial use or profit.

Signature

Date
Physics-based surface energy model optimization for water bodies in cold climates using visible and calibrated thermal infrared imagery

by

May V. Casterline

Submitted to the
Chester F. Carlson Center for Imaging Science
in partial fulfillment of the requirements
for the Doctor of Philosophy Degree
at the Rochester Institute of Technology

Abstract

When tasked with accurately modeling a water body in a cold climate environment, the complexity of the system being simulated and the numerous parameters affecting the observable outcome pose an arduous task for any modeling effort. The task is increasingly complicated when the body of water is serving as a cooling pond for a power plant and can become partially frozen. The introduction of a heat effluent into the water creates a highly dynamic system whose physical state is not only reactionary to the surrounding environmental conditions, but the industrial facility’s operating parameters as well. Both calibrated thermal and visible imagery offer a powerful and unique source of validation data for these hydrodynamic modeling codes when trying to simulate these industrial processes in cold climates. This work presents an approach which uses an evolutionary optimization algorithm to drive the inputs of a hydrodynamic modeling code simulating a power plant cooling pond through imagery validation. The result of this process is an optimized set of functional parameters to the hydrodynamic model that best simulates the observed conditions. While applied to a hydrodynamic code for this work, the process created introduces a unique infrastructure for solving multi-dimensional, multi-system problem sets in a modular and evolutionary framework.
Acknowledgements

There are numerous people, without which, this work would never have been possible. The first person on this long list is my advisor, Carl Salvaggio. Carl taught me how to think critically, how be a better scientist, and to not lower my expectations for myself. He allowed me to go down several rabbit holes of thought before coming out with a good idea, irregardless of the time and frustration it cost. Whether this was engineered or a happy accident, the result is a self-confidence for which I will always be grateful. Al Garrett spent many hours helping me understand thermodynamics, fluid dynamics, and his ALGE code upon which the majority of this work is based. I learned the value of being able to derive ideas from first principles and to stop and think about what you are looking at before making a conclusion. Ed Hensel agreed to not only be on my committee, but also advise my independent studies into thermodynamics and fluid dynamics. He had no idea who I was and I had no idea what I was doing. I thank him for his patience and willingness to explain things to me over and over again.

The two winters spent in Midland, MI are definitely memories I’ll always have. I’d like to thank Brent Bartlett, Jason Faulring, Bob Kremens, Sarah Paul, and Nina Raqueno for all their hard work, manual labor, and time spent in sub-zero temperatures to help with the effort. Despite what legal council may say, snow tubes are perfectly good personal floatation devices when on frozen lakes.

And finally, I’d like to thank my family. I completed this thesis while working full-time, a feat that would not have been possible without the help and support of my parents and siblings. My husband, Brian, has been a constant source of support and encouragement throughout my entire academic career. The hours of writing and coding and venting would have not been possible if he hadn’t been willing and eager to hold down the fort, babysit, and listen.

To my daughter, Claire - Mommy doesn’t have to work on the computer on Saturday any more, we can go to the park!
This work is dedicated to my Dad, who knew I was a scientist before I did.
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract</td>
<td>IV</td>
</tr>
<tr>
<td>Acknowledgements</td>
<td>V</td>
</tr>
<tr>
<td>Dedication</td>
<td>VI</td>
</tr>
<tr>
<td>List of Figures</td>
<td>XIII</td>
</tr>
<tr>
<td><strong>1 Introduction</strong></td>
<td>1</td>
</tr>
<tr>
<td><strong>2 Objective</strong></td>
<td>5</td>
</tr>
<tr>
<td>2.1 Cooling pond basics</td>
<td>5</td>
</tr>
<tr>
<td>2.2 The big picture</td>
<td>6</td>
</tr>
<tr>
<td>2.3 Workflow</td>
<td>6</td>
</tr>
<tr>
<td>2.4 Evolution of thought</td>
<td>7</td>
</tr>
<tr>
<td><strong>3 Theory</strong></td>
<td>9</td>
</tr>
<tr>
<td>3.1 Optimization routines</td>
<td>9</td>
</tr>
<tr>
<td>3.1.1 Hill climbing algorithms</td>
<td>9</td>
</tr>
<tr>
<td>3.1.2 Genetic algorithms</td>
<td>10</td>
</tr>
<tr>
<td>3.2 Particle swarm optimization (PSO)</td>
<td>12</td>
</tr>
<tr>
<td>3.2.1 Basic concepts</td>
<td>12</td>
</tr>
<tr>
<td>3.2.2 Swarm explosion</td>
<td>15</td>
</tr>
<tr>
<td>3.2.3 Example PSO Application</td>
<td>17</td>
</tr>
<tr>
<td>3.3 Summary</td>
<td>21</td>
</tr>
</tbody>
</table>

VII
4 Background

4.1 ALGE hydrodynamic model ........................................... 23
4.2 Operating ALGE ......................................................... 25
4.3 RIT WASP system ...................................................... 28
  4.3.1 WASP Sensor Blackbodies ....................................... 29
4.4 Ice coverage camera .................................................. 30
4.5 Data validation site ................................................... 31
4.6 Summary ................................................................. 32

5 Methodology ............................................................... 33

5.1 Data collection .......................................................... 33
  5.1.1 Autonomous weather station .................................... 34
  5.1.2 Manual ground truth collection .................................. 35
  5.1.3 Ice cover estimation ............................................... 36
  5.1.4 Calibration of WASP imagery .................................... 40
5.2 Evaluation of ALGE simulations ...................................... 41
  5.2.1 Root Mean Squared Error (RMSE) ............................... 41
  5.2.2 Modified Root Mean Squared Error (M-RMSE) ................ 42
  5.2.3 Application to data sets .......................................... 43
  5.2.4 Metric calculation for water temperature ...................... 44
  5.2.5 Metric calculation for ice coverage ............................. 45
  5.2.6 RMS Metric comparison .......................................... 46
  5.2.7 Metric combination ............................................... 50
  5.2.8 Metric weighting .................................................. 50
5.3 Correlation between ice extent and heat load ..................... 52
5.4 Application of PSO to ALGE .......................................... 54
  5.4.1 Optimizing parameter choice .................................... 55
  5.4.2 Temporal input averaging ........................................ 56
  5.4.3 Convergence condition ............................................ 58
  5.4.4 Implementation of PSO-ALGE on computing cluster ........ 58
5.5 Summary ................................................................. 59
6 Results

6.1 Initial optimization results ........................................ 61
   6.1.1 Winter 2008-2009 Results .................................. 63
   6.1.2 Winter 2009-2010 Results ................................. 65
   6.1.3 Overall conclusions from initial simulations .............. 67

6.2 Swarm initialization impact .................................. 70

6.3 Sparse validation data theory ................................. 71
   6.3.1 Sparse validation data experiment ......................... 74
   6.3.2 Sparse validation data results ........................... 75

6.4 Swarm Repeatability ........................................ 85

6.5 Summary .................................................. 86

7 Summary and Conclusions ...................................... 89

7.1 Recommendations for future work .......................... 91

Bibliography .................................................. 95

A Radiation propagation in the thermal region .............. 97
   A.1 Self-emittance .............................................. 97
   A.2 Planck’s equation ......................................... 98
   A.3 Emissivity ................................................ 99
   A.4 Conservation of energy and Kirchoff’s law .............. 100
   A.5 Atmospheric and background effects ....................... 102
   A.6 Atmospheric transmission ................................ 103
   A.7 Sensor-reaching radiance ................................ 104

B Temperature retrieval methods .......................... 107
   B.1 Atmospheric compensation ................................ 107
   B.2 Correlation with ground-based measurements .......... 108
   B.3 Ground-based temperature measurement ................. 109

C Direct temperature retrieval ............................. 111
   C.1 Contact vs. non-contact methods ......................... 111
   C.2 Skin temperature effects ................................. 112
D Error propagation

E WASP sensor calibration
   E.1 Digital counts to radiance ........................................ 117
   E.2 Radiance to temperature .......................................... 119

F Ground instrument calibration ........................................ 125

G WASP sensor sensitivity analysis ..................................... 127

H Imagery Calibration .................................................... 131

I Autonomous in-situ measurements .................................... 137
   I.1 Winter 2008-2009 .................................................. 137
   I.2 Winter 2009-2010 .................................................. 138

J Ice thickness probes ................................................... 143

K Skin temperature experiment .......................................... 147

L On-board blackbody calibration ...................................... 151

M Differential Equations: Sensitivity Analysis ....................... 155
   N Skin Temperature Model ............................................. 157
      N.1 Energy flux through control volumes ......................... 158
      N.2 Control Volume 1: Dry air-water vapor mixture ............ 160
      N.3 Control Volume 2: Gas/water interface ....................... 165
      N.4 Control Volume 3: Pure water ............................... 168

O Swarm Repeatability Results ......................................... 171

P ALGE input files ...................................................... 185
   P.1 param.dat .......................................................... 185
   P.2 idepth.dat .......................................................... 188
   P.3 igrid.dat ............................................................ 189
   P.4 flow.dat ............................................................ 190
Q Computing cluster submission scripts
Q.1 SLURM submission scripts
  Q.1.1 config.sh
  Q.1.2 generation-step.sh
  Q.1.3 submit-generation.sh
Q.2 SGE submission scripts
  Q.2.1 config.sh
  Q.2.2 generation-step.sh
  Q.2.3 submit-generation.sh

R WASP Calibration Code
R.1 WASP_CAL_FILEHANDLER
R.2 CAL_WASP_PREPROCESSING
R.3 WASP_THERM_COMMAND
R.4 COMPUTE_RADIANCE
R.5 COMPUTE_SLOPE_INT
R.6 COMPUTE_STD_LININTERP
R.7 RESAMP_DEADPIX
R.8 READ_ENVI_HEADER
R.9 READ_ENVI_IMAGE

S PSO-ALGE Code
S.1 PSO_CLUSTER
S.2 ALGE_EVAL_ALL
S.3 ALGE_EVAL_FLOW
S.4 ALGE_EVAL_FLOW_2WEEK
List of Figures

3.1 Three dimensional graph of a simplistic function with a single maximum [25] 10
3.2 Three dimensional graph of a more exotic function exhibiting both a local and global maximum [25] 11
3.3 A two dimensional representation of how an individual particle’s current trajectory, personal best solution, and the global best solution all impact the position of the same particle in the next generation. 14
3.4 The velocity update equation for PSO with the individual components identified. 15
3.5 The two-dimensional Rastrigrin function. 17
3.6 Initial swarm distribution for the Rastrigrin swarm. Each point indicates the location of each particle in the solution space at initialization ($t = 0$). 18
3.7 Evolution of the Rastrigrin solutions through the lifetime of swarm population. The population of personal best solutions are shown in black. 19
3.8 Evolution of the Rastrigrin solutions through the lifetime of swarm population. The population of personal best solutions are shown in black. 20
3.9 Evolution of the Rastrigrin solutions through the lifetime of swarm population. The population of personal best solutions are shown in black. 20

4.1 ALGE surface temperature predictions for the Midland Cogeneration Venture cooling pond in Midland, MI using historical weather information. Blue indicates colder temperatures and red indicates warmer temperatures. Solid, dark blue areas represent areas covered with well formed ice. 24
4.2 Data workflow for ALGE comparison process. A user selected combination of inputs generated simulated ice, temperature, and flow velocities that are then compared to calibrated thermal imagery and observed ice cover fractions for validation. ................................................................. 25

4.3 WASP sensor in laboratory. ................................................................. 28

4.4 Shown on the left in Figure 4.4(a) are the blackbody reference sources that are mounted on the WASP sensor head. Shown on the right in Figure 4.4(b) is an AutoCAD rendering of the WASP sensor head with the two blackbody reference sources mounted. ................................................................. 30

4.5 Aerial view of MCV facility and cooling pond. Power facility is located on the northern end of the pond with a concrete berm separating the hot out flow on the left from the cold intake on the right. ................................................................. 31

5.1 Weather station constructed on the northern shore of the MCV cooling pond 34

5.2 Photos from ground truth campaign. Figure 5.2(a) shows a team member measuring water temperatures with a thermistor. Figure 5.2(b) shows the airboat used for data collection. ................................................................. 35

5.3 Original image captured from rooftop with processed version with geometrical distortions from fisheye lens removed. It should be noted that pixels near the edges of image begin to exhibit correction errors. This is due to low point density of calibration grid at the edge of the FOV. ................................. 37

5.4 Ice and water pixels have been identified with two different ROI’s. Water is tagged as blue, ice as red. ................................................................. 37

5.5 Tangent line needed for calculations of scale shown in 5.5(a). Length $OK'$ and the angle of the tangent line relative to the X-axis are needed. Side view of principle plane of the oblique photograph shown in 5.5(b). ................... 38

5.6 Images used in comparison and error metric calculations. Left-most image shows de-fished fisheye images. Middle images show nadir captured LWIR images used to select the open water ROIs for error comparison. .................... 40

5.7 WASP image collected 10 March 2010 with collected temperature points overlaid. ................................................................. 44

5.8 Comparison of simulated and measured ice fractions using measured environmental and plant data from the 2008-2009 winter. ................................................................. 47

XIV
5.9 Metric sensitivity to 2008-2009 data set. $RMS_{mod,ice}$ and $RMS_{mod,water}$ represent the metric data using the modified RMS method while $RMS_{norm,ice}$ and $RMS_{norm,water}$ represent the metric data using the traditional RMS calculation. 

5.10 Metric sensitivity to 2009-2010 data set. $RMS_{mod,ice}$ and $RMS_{mod,water}$ represent the metric data using the modified RMS method while $RMS_{norm,ice}$ and $RMS_{norm,water}$ represent the metric data using the traditional RMS calculation.

5.11 Comparison of average simulated ice fraction to waste heat load. Each point represents the average simulated ice fraction for an entire winter simulation.

5.12 Comparison of average difference between average water temperature and air temperature to waste heat load. Each point represents the average difference between the two temperature values for an entire winter simulation.

5.13 Comparison of temporal ice fractional coverage for each heat load condition.

5.14 Comparison of average heat load and average ice fractional coverage for each heat load condition.

5.15 High resolution flow rate (black) plotted with the average flow rate (red) using an approximate 6-day window. This average flow rate plotted here is not a result of the PSO process. This data is included to help illustrate the temporal averaging implementation and implications.

5.16 Graphical representation of the PSO optimized ALGE algorithm as implemented on a computing cluster.

6.1 Correlation between the observed and simulated ice fractions for the best particle in the first and final generations of the 2008-2009 winter simulations.

6.2 Comparison of the averaged flow rate used to produce both the initial and final simulations for the swarm’s best solution. The true average, derived from known plant parameters, is shown in blue while the swarm results are shown in red. And ideal solution would produce an averaged flow rate matching the true average.

6.3 Correlation between the observed and simulated ice fractions for the best particle in the first and final generations of the 2009-2010 winter simulations.
6.4 Comparison of the averaged flow rate used to produce both he initial and final simulations for the swarm’s best solution during the 2009-2010 winter. The true average, derived from known plant parameters, is shown in blue while the swarm results are shown in red. And ideal solution would produce an averaged flow rate matching the true average. ............................. 67

6.5 Comparison of all the flow rates used to produce both the initial and final simulations for the entire swarm during the 2008-2009 winter. The true average, derived from known plant parameters, is shown in blue while the swarm results are shown in red. And ideal solution would have all the swarm produced flow rates converging to match the true average flow rate. 68

6.6 Comparison of all the flow rates used to produce both the initial and final simulations for the entire swarm during the 2009-2010 winter. The true average, derived from known plant parameters, is shown in blue while the swarm results are shown in red. And ideal solution would have all the swarm produced flow rates converging to match the true average flow rate. 69

6.7 Initial and final flow rate distribution for the trial swarms for the 2009-2010 data set. This swarm was initialized at a lower mean flow rate and a smaller variance to help mitigate the initialization failures of the previous full winter simulations. ................................. 70

6.8 Comparison of initial and final ice fraction correlations for the re-run 2009-2010 swarm. This swarm was initialized at a lower mean flow rate and a smaller variance to help mitigate the initialization failures of the previous full winter simulations. ................................. 71

6.9 Initial and final flow rate distribution for the trial swarms for the 2009-2010 data set with a sparser validation data set. ................................. 72

6.10 Comparison of initial and final ice fraction correlations for the sparse validation data swarm. ................................. 73

6.11 Comparison of initial and final ice fraction correlations for the 1-hour validation interval. ................................. 75

6.12 Comparison of initial and final ice fraction correlations for the 12-hour validation interval. ................................. 76

6.13 Comparison of initial and final ice fraction correlations for the 24-hour (1 day) validation interval. ................................. 76

XVI
6.14 Comparison of initial and final ice fraction correlations for the 48-hour (2 day) validation interval. ......................................................... 77
6.15 Comparison of initial and final ice fraction correlations for the 72-hour (3 day) validation interval. ......................................................... 77
6.16 Comparison of initial and final ice fraction correlations for the 120-hour (5 day) validation interval. ......................................................... 78
6.17 Comparison of initial and final ice fraction correlations for the 168-hour (7 day) validation interval. ......................................................... 78
6.18 Comparison of initial and final ice fraction correlations for the 288-hour (12 day) validation interval. ......................................................... 79
6.19 Comparison of initial and final ice fraction correlations for the 576-hour (24 day) validation interval. ......................................................... 79
6.20 Swarm flow rates for constant flow rate conditions and 1-hour validation interval. .............................................................. 80
6.21 Swarm flow rates for constant flow rate conditions and 12-hour validation interval. .............................................................. 80
6.22 Swarm flow rates for constant flow rate conditions and 24-hour (1 day) validation interval. .............................................................. 81
6.23 Swarm flow rates for constant flow rate conditions and 48-hour (2 day) validation interval. .............................................................. 81
6.24 Swarm flow rates for constant flow rate conditions and 72-hour (3 day) validation interval. .............................................................. 82
6.25 Swarm flow rates for constant flow rate conditions and 120-hour (5 day) validation interval. .............................................................. 82
6.26 Swarm flow rates for constant flow rate conditions and 168-hour (7 day) validation interval. .............................................................. 83
6.27 Swarm flow rates for constant flow rate conditions and 288-hour (12 day) validation interval. .............................................................. 83
6.28 Swarm flow rates for constant flow rate conditions and 576-hour (24 day) validation interval. .............................................................. 84

A.1 Blackbody energy distributions for blackbodies sitting at 5800K, 3000K, and 300K. A solar spectra is compared to the energy distribution of a blackbody sitting at 5800K [22]. ......................................................... 99
A.2 Atmospheric transmission spectrum. Regions of the atmosphere exhibiting 0% transmission are optically opaque because the corresponding molecule absorbs energy at the specified wavelengths. [26].

A.3 Thermal energy paths from potential sources of interest. Path A photons are photons emitted by the atmosphere and radiated directly into the sensor. Path B photons are emitted from some background object, fall onto the target, and are then reflected back towards the sensor. Path C photons are photons emitted by the target of interest. Path D photons are emitted by the atmosphere, towards the target, reflect of its surface, and are collected at the sensor.

C.1 Idealized temperature profiles of the near-surface layer of a water body. Blue line represents the temperature profile of a body water during the day in low wind conditions. The purple, dotted line represents the temperature profile of the same body of water during the night or day in strong wind conditions.

E.1 Flight line collection scheme for blackbody imagery collection during flight. Green squares represent frames over the area of interest. Red and blue squares represent frames acquired while the blackbody is in the field of view of the LWIR detector and set to a hot and cold set point, respectively.

E.2 Digital count to radiance conversion depicting the linear relationship assumed between the digital counts recorded at each pixel and the apparent radiance emitted from the blackbody at each temperature set point.

E.3 Relationship between radiance and temperature for entire range possible scene temperatures as well as a fitted linear model.

E.4 Relationship between radiance and temperature for subset of thermal range of possible scene temperatures as well as a fitted linear model.

E.5 Graphic depicting apparent sensor-reaching radiance to temperature conversion. Each point, both shown spatially in the image domains and graphically in the plot, represent points on the ground where geo-located temperature measurements were recorded. The relationship between the calculated sensor-reaching radiance at these points and their corresponding temperatures measured on the ground is then fit with a linear model.
F.1 Calibration plot for skin temperature retrieval instruments generated by data collected in the field using a blackbody thermal source. 126

G.1 Graphic illustrating the calculation of the variance terms for each pixel. The variances are calculated for each pixel using the average of each series of blackbody images. These calculated variances are then regressed against their corresponding digital counts to determine the variance gain and bias masks. 129

H.1 Calibrated temperature map for the 11 February 2010 night collect created using the implemented calibration technique. Temperature is reported in Celsius. 132

H.2 Calibrated temperature map for the 4 March 2010 night collect created using the implemented calibration technique. Temperature is reported in Celsius. 132

H.3 Radiance to ground temperature calibration model for the 11 February 2010 night collect. Each point represents a geo-located temperature measurement and its corresponding sensor-reaching radiance. 133

H.4 Radiance to ground temperature calibration model for the 4 March 2010 night collects. Each point represents a geo-located temperature measurement and its corresponding sensor-reaching radiance. 134

I.1 Inside of buoy containing datalogger, multiplexors, GPS units, battery, and cellular modem shown in I.1(a). Ice thickness probe (long, black pole) mounted on side of buoy shown in I.1(b). 138

I.2 Buoys in both open water and iced conditions while deployed in the MCV cooling pond during the 2008-2009 winter collection season. 139

I.3 Comparison of buoy design differences between the 2008-2009 and 2009-2010 winter collection seasons. 140

I.4 Newly designed buoys in the RIT laboratory 141

I.5 Newly designed buoys afloat in MCV cooling pond 141

J.1 Inside of buoy containing datalogger, multiplexors, GPS units, battery, and cellular modem shown in J.1(a). Ice thickness probe (long, black pole) mounted on side of buoy shown in J.1(b). 143
Chapter 1

Introduction

The marriage of remotely sensed imagery and three-dimensional hydrodynamic models yields a very powerful tool for simulating and inspecting complicated water environments. Traditionally, thermal information extracted from calibrated infrared imagery is the validation data for a hydrodynamic simulation and a source of comparison for evaluating the accuracy of the predicted environments. The complexity of the relationship between reliable validation imagery and the modeling environment is dramatically increased in cold climates. Water environments exposed to below freezing conditions present a challenge to both standard thermal calibration techniques as well as traditional hydrodynamic models. Any water present can exist in several different states simultaneously within an area: liquid water, solid ice, or snow. The thermal variation and distribution of water, ice, and snow is dependent on the inherent physical characteristics of the body of water as well as the meteorological conditions at any given time. It is very difficult to accurately describe the thermal and physical parameters of such an area (also referred to as a scene) using direct measurement techniques due to the spatially and temporally varying nature of said parameters.

Given some preliminary knowledge of the physical parameters bounding the observed scene, three-dimensional hydrodynamic models can calculate the energy fluxes at the observed surface and provide estimations for physical parameters. However, this powerful modeling tool can quickly become cumbersome if only limited knowledge of the environmental conditions is available. Lack of environmental knowledge leads to a “guess and check” approach in trying to select model inputs capable of generating results that adequately compare to environment observations. Additionally, the spatial variation evident
in frozen water bodies, paired with the environmental challenges introduced when operating in cold climates, creates a situation where empirically measuring all data necessary for total model validation is a physical impossibility. With limited empirical measurements, only so many system assumptions can be made before the accuracy of a simulation code is questionable.

Both remote broadband thermal mapping and traditional remote sensing (VNIR) approaches, offer unique data collection capabilities for cold climate environments. Thermal mapping techniques provide instantaneous spatial coverage of extended areas and capture the varying thermal structure of the scene. The sensitivity of a broadband thermal sensor is limited to extracting thermal radiance properties of only the scene’s surface and sheds no light on the three-dimensional physical structure below. Additionally, access to consistent thermal coverage of an area of interest is often not possible. Traditional VNIR technologies capture the same instantaneous spatial coverage as a thermal sensor, often at better spatial resolution, but cannot provide the same insight into the thermodynamic conditions on the ground. However, airborne or space-based VNIR systems with adequate spatial resolution are capable of observing the amount of ice coverage on a body of water which is a valuable observable in cold climate conditions. VNIR imagery is often more accessible and can offer more consistent observations.

The research presented here uses a genetically-inspired optimization algorithm (particle swarm optimization [17]) to drive the selection of input parameters for a three-dimensional hydrodynamic model (ALGE [9]) while using image-derived information for model validation. The operational goal of ALGE is to simulate a cooling pond for an industrial facility in a cold climate environment. In order to determine the level of success the model achieved when driven by this method, calibrated thermal imagery, airborne VNIR imagery, and other remotely sensed imagery of the actual water body being modeled is compared to the simulated state of the same scene. Assuming a favorable comparison, a simulation is considered successful if the corresponding input ALGE parameters effectively model the facility’s heat load within 10% of the true value [10]. To restrict the potential solution inside the boundaries of physical possibilities, the solution space to be searched is defined by physics-based estimates, derived from empirical measurements.

The employment of an optimization routine in parameter selection has the potential to increase the simulated data’s accuracy as well as decrease the time required to produce a quality simulation. Because the optimization is a genetic algorithm and evolutionary in
nature, only the necessary amount of simulations will be performed to achieve a defined
goal. In comparison to a brute-force technique where every possible parameter combination
is run to create a large look-up table of simulations, a genetic algorithm conserves valuable
computational time, performing only necessary simulations until a satisfactory result is
achieved. This implementation has the potential to save hundreds of hours in simulation
time in order to achieve the same results.

The overall contributions from the presented work are listed below. Each of these
contributions are explained in detail in the following chapters.

- Developed and implemented automated calibration system for the WASP sensor
  system. The implemented technique includes a flight line specific thermal radiance
  calibration on a pixel-by-pixel basis.

- Created a general-use framework for model-wide optimization which uses imagery-
  derived information as a validation metric. While the specific model and algorithm
  used here were ALGE and PSO, the approach created can be applied to any type of
  multi-dimensional system.

- Implemented an evolutionary optimization framework currently operational on a
  multiple core computing cluster. The system design was intentionally created in a
  modular fashion to allow the system to be applicable to other problem sets.

- Developed an approach to using temporally varying data as optimization inputs in
  a way that is both feasible and simple to implement.

- Demonstrated a strong correlation between observed ice fraction and the heat load
  present in a cooling pond. This correlation allows observed ice fractions to be the
  only necessary validation data when optimizing the ALGE routine. The implication
  of this relationship is a simple and elegant method for model validation and allows
  the approach to be executed when only visible imagery is available.

- Evaluated and determined an adequate collection interval for validation imagery
  observations. These results have a practical impact on the operational requirements
  of an application of this technique.

- Managed and executed two winter ground-truth campaigns to support the validation
  of the ice modeling capabilities added to the ALGE model. These campaigns resulted
in a library of meteorological, plant conditional, and multi-modal image data from both winters.

• Wrote a suite of IDL tools for interfacing with the ALGE model. These tools allow a user to extract results from specific points in time with minimal interaction with the raw data. Additionally, tools were created for evaluating an ALGE simulation with imagery in a headless and automatic fashion.

• Created a set of analyzing tools to inspect the progress of a given optimization in real-time. These tools aid the user in determining if the optimization is behaving as expected.
Chapter 2

Objective

The ultimate goal of this work is to improve upon the accuracy of the ALGE modeling environment when simulating industrial cooling water bodies in cold climates. The improvement is achieved through the use of image-derived information as validation metrics in an intelligent and efficient manner. The intelligent manner in this case is the implementation of particle swarm optimization (PSO) to perform the input parameter selection for ALGE. The following chapter explores how all the systems involved are necessary and linked together to form the final process workflow. A more detailed background on each of the systems is developed in Chapter 4.

2.1 Cooling pond basics

Cooling ponds are used by industrial facilities as a means of waste heat dissipation. The passive nature of this method of cooling creates a very reliable system that requires no maintenance or user interaction. A cooling pond is a body of water, either man-made or natural, within which a facility will inject its waste heat via hot water. The system will then replace the lost volume with cooler water from some extraction point located at a separate position, some distance from the injection site. To maximize the cooling efficiency of the system, the injection and intake points are often separated by some type of division, whether it be a manmade berm or natural barrier. An ideal cooling pond will have a large enough surface area to allow the hot effluent to cool as to close to ambient temperature as possible.

Typically, the size of cooling ponds is designed or chosen so the body of water will
be resistant to freezing during the colder months. By definition, there is a constant heat injection into the body of water; provided the size of the pond is sufficiently small, the cold weather will not be able to induce freezing. However, these ideal conditions are not always the case. When a facility is not dispensing a large enough heat load into its pond or the pond is too large, ice can form on the water’s surface. An ice layer acts as an insulating blanket and reduces the surface area of open water exposed to the air. As conductive cooling is a significantly less efficient method of cooling, when compared to convective and evaporative, ice formation adversely affects the cooling efficiency of a pond. Any sort of additional snow accumulation will further exacerbate the problem. As a result of all these complexities, an already difficult environment to encapsulate in numerical models is made even more challenging with the introduction of the cold weather phenomenology.

2.2 The big picture

While recent research has extended the ALGE model’s capability to simulate ice and snow formation on a cooling pond [12], the problem at hand still remains complex. In reality, when the model is applied to a real-world situation, the likelihood of having a well characterized area with access to all meteorological, environmental, and facility operational data is slim. However, there is a higher probability a user can access either remotely sensed thermal or visible imagery. The imbalance in available information is what leads to the approach implemented in this work. The methodology put forth and tested provides a user with a process to methodically determine adequate model parameters that accurately simulate a cooling pond in a cold environment. The workflow is developed in the next section.

2.3 Workflow

The overall workflow for the process is as follows:

1. Collect the appropriate calibrated, empirical data for use in model validation.
   - This data comes from imagery sources and data collection campaigns.

2. Designate the bounding conditions for the water body of interest based on existing knowledge about the environment. These bounding conditions include the
bathymetry and the physical boundaries.

3. Decide which ALGE parameters are going to be variables in the optimization.
   - Possibilities include the flow rate, plant temperature differential over the corresponding simulation time, the meteorological conditions, or all of the above.
   - The choice of variable is based on which information the user has the most confidence in versus which set is considered less reliable.

4. A PSO swarm of ALGE simulations, conforming to the the boundary and initialization conditions, is initialized on a computing cluster.

5. In the swarm, each solution (or particle) is evaluated using the validation metric to compare the empirically known data to the offered simulated data.

6. The PSO architecture drives the selection of subsequent generations of ALGE input parameters based on the successes and failures of previously offered configurations.

7. Optimization continues until a user-chosen convergence parameter is met or the allotted number of PSO generations complete.

8. Depending on the outcome, either the converged solution or the best solution achieved at the end of the swarm’s lifetime, are designated as the optimized, best set of input parameters for ALGE to model the given environment.

2.4 Evolution of thought

Based on previous work and experience, the initial focus of this work was on the potential dependency on thermal data as a validation source. It was hypothesized that due to the incredibly complicated state of the cooling pond environment at any given time, insight into the current thermodynamic conditions on the ground would be necessary to accurately model the system. As a result, significant efforts were invested into the thermal collection techniques surrounding the two executed ground truth campaigns, as well as thermal phenomenology. These efforts included the implementation of automated calibration routines for a broadband, airborne thermal sensor, subsequent sensitivity analysis of calibration results, construction and deployment of autonomous buoys to monitor thermal conditions at the ground truth site, and research into thermodynamic principles and phenomenology. As the work evolved it became evident that the emphasis on the thermal information was
unnecessary and attention was shifted to a more accessible and pragmatic observable: the overall extent of the cooling pond covered with ice at given point in time. With this modification of focus, a large portion of the collected data on the ground became immaterial to the final experiments. The progression to this final conclusion and the resulting experiments is explained in the following chapters. However, the significant amount of work that ultimately led to the final experiment and solution remains documented in the appendices.
Chapter 3

Theory

At the core of this work is the application of an exotic optimization routine to the ALGE model. This chapter will review the basic principles of optimization. The areas covered will include different optimization approaches, with a focus on particle swarm optimization. As referred to in Section 2.4, extensive research was completed on thermal phenomenology. While not included in this chapter, the work is documented in Appendices A-D.

3.1 Optimization routines

The goal of an optimization routine is to determine the minimum or maximum of some real function through a systematic search of a range of possible solutions. An optimization approach becomes advantageous to a particular problem when the potential solution space is so large that multiple results need to be compared to determine the best solution. Two types of optimization approaches are described below: hill climbing techniques and genetically-influenced techniques.

3.1.1 Hill climbing algorithms

Hill climbing optimization routines are forms of local search and encapsulate more traditional search techniques. In general, these approaches can be used to solve problems that have multiple solutions. Initially a potential solution is chosen and then alterations are made iteratively to the solution to generate a neighborhood solution. If the neighborhood solution is better than the current solution then it becomes the new current solution. This
process continues until the current solution cannot be improved. For example, take the function \( f(x, y) = e^{-(x^2 + y^2)} \). This particular function has a single maximum value at \([0, 0]\) as seen in Figure 3.1. Each vertex on this plot represents a possible solution. In order to determine the optimal maximum solution, the hill climbing routine will move through the function, vertex to vertex, locally increasing the functional evaluation of \( f \) until it has reached the peak.

![Figure 3.1: Three dimensional graph of a simplistic function with a single maximum](image)

Hill climbing approaches are relatively simple optimization routines to implement and, as a result, are popular choices \([? ]\). However, because hill climbing relies on making discrete steps to neighboring values in the solution space, it is often possible to obtain a locally optimal solution instead of the globally optimal solution for a complex function. As shown in Figure 3.2, a more complex function having both a local and a global maximum would present a challenge to this simplistic approach. In addition, any plateaus present in a solution space would pose a problem to this search method. Because neighboring values would be nearly indistinguishable in these areas, the algorithm will tend to wander aimlessly inside the plateau and cease any improvement. Additionally, because these approaches are single-node methods, it can take a considerable amount of time to effectively traverse a solution space. This characteristic becomes particularly troublesome when the function being optimized is computationally expensive.

### 3.1.2 Genetic algorithms

Genetic algorithms belong to a larger class of evolutionary algorithms, all of which generate solutions using techniques inspired by natural evolution processes such as inheritance, mutation, selection, and crossover \([20]\). These approaches become advantageous when
3.1. Optimization routines

If this type of approach was applied to the function shown in Figure 3.1 several vertices would originally be chosen and comprise the first generation of the algorithm. Each vertex would be evaluated and the one yielding the largest functional value would be considered the best solution. All or some of the solutions would then be altered to position them at different vertices and then compared to the best solution from the previous generation. This process would complete itself when the maximum value is determined or the maximum number of generations allowed have been created.

Generally, genetic algorithms have been developed and used without an established theory as to why they perform well or why they fail in certain situations. As a result, the parameters which influence the performance of a given algorithm tend to be untested and unknown for a particular implementation. In addition, it is difficult to describe the evolution of a given population, which leads to challenges in altering inputs for improved performance. However, genetic algorithms perform well in solution spaces that are large, multi-dimensional, and contain many hills and valleys. For example, a genetic algorithm would perform well with the complex solution space presented in Figure 3.2. Unlike the hill

\[ f(x,y) = e^{-\frac{(x^2+y^2)}{2}} + 2e^{-((x-1.7)^2+(y-1.7)^2)} \]

Figure 3.2: Three dimensional graph of a more exotic function exhibiting both a local and global maximum [25]
climbing approach whose single investigation location can get stuck in the local maxima, a genetic algorithm would have several solutions instantiated across the solution space. Even if a single solution was trapped in the local maxima, there would be several other individual solutions left to explore the remainder of the solution space.

3.2 Particle swarm optimization (PSO)

Particle swarm optimization, or PSO, is a genetic algorithm introduced by Eberhart and Kennedy in 1995 which instantiates several solutions simultaneously to search multidimensional solution spaces [17]. Inspired by the social behavior of birds, PSO mimics the flock behavior that birds demonstrate while searching for food. If one imagines a flock of birds circling above some location, the ultimate goal of the flock is to find food somewhere on the ground below. If a single bird discovers food and dives to the ground, the entire flock will shift their individual courses and follow the bird diving to the food. The behavior of each member in the flock is influenced by both its own history of success and the success of the other members of the flock.

3.2.1 Basic concepts

By extension, the flock behavior can be applied to solution optimization. In PSO, a potential solution to a function is represented by a particle (analogous to a bird). A group of possible solutions, or particles, is referred to as a swarm (analogous to a flock). Each particle is defined by a set of parameters. These parameters are initially unique to each particle and are the variables of the function to be optimized. A swarm is composed of N particles (solutions) that each have J-number of parameters. Each ith particle, \( \vec{x}_i [\rho_1, \rho_2, ..., \rho_J] \) for \( i = 1, ..., N \), is initialized with random parameter values, bounded by each parameter condition. The quality of each particle’s offered solution is judged by computing a fitness function using the parameters associated with that particle. For each iteration, or generation, the outcome of the function evaluation for each particle is tracked. A particle’s best solution, \( \vec{P}_i \), is the set of parameters from that particle’s history that have produced the best fitness function value to date. The global best solution, \( \vec{G} \), is the parameter set that has produced the best functional value obtained out of the entire population to date. Using these two parameter sets, as well as other motion parameters, a velocity vector (Equation 3.1) for each particle is calculated.
3.2. Particle swarm optimization (PSO)

\[
\vec{v}_i(t + 1) = \vec{v}_i(t) + \alpha_p(t) \gamma_p \left( \vec{P}_i - \vec{x}_i(t) \right) + \alpha_g(t) \gamma_g \left( \vec{G} - \vec{x}_i(t) \right)
\]  

(3.1)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\vec{v}_i(t + 1)$</td>
<td>current velocity vector of the $i$th particle</td>
</tr>
<tr>
<td>$\vec{v}_i(t)$</td>
<td>previous velocity vector of the $i$th particle</td>
</tr>
<tr>
<td>$\vec{x}_i(t)$</td>
<td>the position of the $i$th particle</td>
</tr>
<tr>
<td>$\alpha_{p,g}(t)$</td>
<td>weighted stochastic variables between $[0,1]$</td>
</tr>
<tr>
<td>$\gamma_p$</td>
<td>cognitive acceleration constant</td>
</tr>
<tr>
<td>$\gamma_g$</td>
<td>social acceleration constant</td>
</tr>
<tr>
<td>$\vec{P}_i$</td>
<td>$i$th’s particle personal best parameter set</td>
</tr>
<tr>
<td>$\vec{G}$</td>
<td>global best parameter set achieved</td>
</tr>
<tr>
<td>$i$</td>
<td>current particle</td>
</tr>
<tr>
<td>$t$</td>
<td>current generation</td>
</tr>
<tr>
<td>$J$</td>
<td>total number of parameters defining a particle</td>
</tr>
<tr>
<td>$T$</td>
<td>total number of generations allowed</td>
</tr>
<tr>
<td>$N$</td>
<td>total number of particles in a swarm</td>
</tr>
</tbody>
</table>

Table 3.1: PSO velocity vector equation variables

The calculated velocity vector for a particle represents the incremental changes applied to each parameter, for a given particle, to move that particle closer to the optimum solution in the solution space. The parameters for each particle in the next generation ($t + 1$) are calculated by adding the velocity vector to the current particle’s parameter set for the current generation, $t$ ($t = 1, \ldots, T$). The position update equation is shown in Equation 3.2. The movement of the swarm continues until all of the personal best solutions obtained by each particle achieves a minimum or maximum functional evaluation within some margin of error, or until the total number of generations allowed, $J$, have occurred.

\[
\vec{x}_i(t + 1) = \vec{x}_i(t) + \vec{v}_i(t + 1)
\]  

(3.2)

An example of an application of PSO in a two-dimensional solution space is shown in Figure 3.3. This figure demonstrates how the current particle’s position, $\vec{x}(t)_i$, it’s personal best solution, $\vec{P}_i$, and the global best solution, $\vec{G}_i$, contribute to the particle’s position in the solution space for the next generation, $\vec{x}(t + 1)_i$. The incorporation of the global and personal best parameters will push all of the particles in the swarm to follow the one
particle achieving the best answer.

Figure 3.3: A two dimensional representation of how an individual particle’s current trajectory, personal best solution, and the global best solution all impact the position of the same particle in the next generation.

The PSO velocity equation can be dissected into three main components, as distinguished in Figure 3.4. The first component is referred to as the inertia component. This term models the tendency of a particle to continue on the same path. The second term is the cognitive term and is sometimes referred to as the particle’s memory, self-knowledge, or remembrance. It represents the particle’s linear attraction towards its personal best solution achieved to date. The third term is called the social term and is sometimes referred to as the social knowledge, group knowledge, or cooperation term. It represents the particle’s linear attraction towards the best position achieved by any particle in the swarm.
3.2. Particle swarm optimization (PSO)

3.2.2 Swarm explosion

As PSO has evolved, several methods have been proposed to facilitate convergence and combat an observed phenomena called swarm explosion. Because of the stochastic nature inherent to the velocity calculation, research has indicated that swarms have a tendency to diverge, sending all particles to infinity (or the solution space boundaries), if not controlled. Swarms can be controlled by several techniques including, limiting the maximum velocity allowed, boundary conditions, selecting appropriate acceleration constants, or an inertia weight. There are other solutions to mitigate swarm explosion, however, for this work these were the four methods chosen to explore [7] [? ].

A maximum velocity sets a limit on how large of a step a given particle can make in a single iteration. This restriction prevents the velocity vector from becoming so large that the particle takes on an uncontrolled trajectory, bouncing from one end of the solution space to another. To prevent solution space wide oscillations, the maximum velocity is applied to restrict the step size, as shown below.

\[
\text{if } \vec{v}[\rho_j]_i > v_{\text{max}} \text{ then } \\
\quad \vec{v}[\rho_j]_i = v_{\text{max}} \\
\text{else if } \vec{v}[\rho_j]_i < -v_{\text{max}} \text{ then } \\
\quad \vec{v}[\rho_j]_i = -v_{\text{max}} \\
\text{end if}
\]

There are several ways to calculate this maximum velocity. Some methods have proposed that the maximum velocity be set to the range of the domain you are searching (Equation 3.3). However, for the research here, a more conservative approach was taken, and the maximum velocity was set to 15% of the domain range[23].

\[
\vec{v}_i(t + 1) = \vec{v}_i(t) + \alpha_p \gamma_p (\vec{P}_i - \vec{x}_i(t)) + \alpha_g \gamma_g (\vec{G} - \vec{t}_i(t))
\]

Figure 3.4: The velocity update equation for PSO with the individual components identified.
\[ v_{max} = 15\% \frac{(x_{max} - x_{min})}{N} \]  

(3.3)

Boundary conditions restrict particles from escaping the solution space. Based on user-defined dimensional limits, if a particle’s position moves outside the boundaries of these dimensions, then it is reverted to the limit value. The logic to make this decision is shown below [19].

\[
\text{if } \vec{x}[\rho_j]_i > x[\rho_j]_{max} \text{ then} \\
\quad \vec{x}[\rho_j]_i = x[\rho_j]_{max} \\
\text{else if } \vec{x}[\rho_j]_i < x[\rho_j]_{min} \text{ then} \\
\quad \vec{x}[\rho_j]_i = x[\rho_j]_{min} \\
\text{end if}
\]

The acceleration constants control the attraction each particle has to the best positions. Smaller values weaken the influence of the social and cognitive terms and lead to limited particle motion. Under these conditions it will take more iterations for the particles to reach an optima. Larger acceleration values can contribute to particle divergence and result in the step size for a given iteration defaulting to the maximum allowable velocity, \( v_{max} \). Several studies have explored the effect these constants have on swarm behavior [6].

Previous research observed that increases in the acceleration constants cause an increased frequency in the oscillations around a solution space’s optimal point. Additionally, when the summation of the two acceleration constants increases beyond 4, the particle trajectories go to infinity. Based on these studies, a ceiling value of 4 is assigned to the summation of cognitive and social acceleration constants. The nature of the problem being optimized will dictate whether these values are held equal or are different. This decision is based on user experience and intuition.

The final tactic for limiting the swarm is the implementation of an inertia weight, \( \phi(t) \) [23]. This parameter is a multiplier of the previous velocity, \( v_i \), in the velocity equation. The inertia weight can be set to a constant, however having the value be a linearly decreasing quantity allows the parameter to control the exploration of the solution space. Initially, with the weight set to a high value (typically \( \approx 1 \)) the particles have freedom to move through the entire space. This phase is referred to as the exploration phase. As time progresses, this factor reduces to 0 at constant increment to decrease the influence of the stochastic velocity and allow the swarm to focus on the personal and global
optimas. This secondary phase is called the exploitation phase. With the addition of the inertia weight factor, the new velocity vector changes to Equation 3.4.

\[
\vec{v}_i(t + 1) = \phi(t)\vec{v}_i(t) + \alpha_p \gamma_p \left( \vec{P}_i - \vec{x}_i(t) \right) + \alpha_g \gamma_g \left( \vec{G} - \vec{x}_i(t) \right)
\]  

(3.4)

### 3.2.3 Example PSO Application

To illustrate how PSO works in application, a swarm was used to solve for the global minimum of the Rastrigin function. The Rastrigin function is a non-convex function that is typically used for performance testing optimization approaches [2]. It presents an optimization approach with a fairly difficult challenge due to the large number of local minima over the extent of the search space.

![Figure 3.5: The two-dimensional Rastrigrin function.](image)

The Rastrigrin function shown in Figure 3.5 is defined by Equation 3.5 where \( A = 10 \) and \( x_i \in [-5.12, 5.12], y_i \in [-5.12, 5.12] \). The global minimum occurs at \([x, y] = [0, 0]\) where \( f[x, y] = 0 \).

\[
f[x, y] = 2A + \left( x^2 - A \cos(2\pi x) \right) + \left( y^2 - A \cos(2\pi y) \right)
\]  

(3.5)
To solve this function, a swarm of 16 particles was generated using the same code engine designed to implement the PSO-ALGE framework. Each particle was initially defined by random parameter values bound by the extent of the solution space. Shown in Figure 3.6 is a 2D view of the initial swarm distribution for the Rastrigin function. Each point represents the initial position of a particle. The parameters for a particle are the guessed $x$ and $y$ values. The swarm was given 250 generations to converge to a point where every particle’s solution was within an error margin of $\pm 1 \cdot 10^{-4}$ of 0. This determination is made by inserting each particle’s parameter set into Equation 3.5, determining the value and comparing it to 0. The smaller the difference between the particle’s value and 0, the better ranked that particle becomes against the global population. It should be noted that for the PSO-ALGE implementation the convergence condition has no known minimum like the Rastrigrin. As a result convergence was determined to have occurred when all the particles were within an error margin away from one another.

![2D Rastigrin Function with swarm at GEN=000](image)

Figure 3.6: Initial swarm distribution for the Rastrigrin swarm. Each point indicates the location of each particle in the solution space at initialization ($t = 0$).

The Rastrigrin swarm successfully converged on the global minimum after 225 gen-
erations. Figures 3.7(a) through 3.9(b) show the evolution of the swarm through time from the perspective of the personal best solution achieved by each particle. A critical component of PSO is that while a particular particle explores the solution space, a social and cognitive memory exists to generate a convergence condition. This behavior is an important feature for PSO in that it ensures that a memory of good candidate solutions are preserved while the particles are still exploring the solution space. Additionally, this feature helps mitigate the swarm succumbing to local minima. These plots show that as time progresses the swarm evolves and the particles converge on the appropriate global minimum.

Figure 3.7: Evolution of the Rastrigrin solutions through the lifetime of swarm population. The population of personal best solutions are shown in black.

From the point of view of this example function, the particle/parameter relationship and how it relates to the functional output is easily understood. When applied to the PSO-ALGE framework, the implementation is not as straightforward. For reasons explained in Sections 5.4.1 and 5.4.2, only a single parameter is being optimized from the ALGE perspective. However, the implementation of a temporal input averaging technique translates the single ALGE parameter into a several parameters in the temporal dimension from the PSO perspective.
Figure 3.8: Evolution of the Rastrigrin solutions through the lifetime of swarm population. The population of personal best solutions are shown in black.

Figure 3.9: Evolution of the Rastrigrin solutions through the lifetime of swarm population. The population of personal best solutions are shown in black.
3.3 Summary

This chapter reviewed the fundamentals of the optimization technique utilized for this work. The mathematical basis for the optimization routine, particle swarm optimization, was explored. The next chapter will discuss, in specifics, the multiple systems which had to be used to construct the approach utilized.
Chapter 4

Background

This chapter provides an overview of the different systems that are required to cooperate with one another in order to determine the validity of the developed approach. These systems included the ALGE hydrodynamic code, Rochester Institute of Technology’s WASP instrument, and an oblique imaging system. Enhancements and modifications were required of all systems in order to employ them in this work. Additionally, the data validation site is discussed.

4.1 ALGE hydrodynamic model

ALGE is a three-dimensional (3D) hydrodynamic model developed at the Savannah River National Laboratory as part of the Multi-spectral Thermal Imager (MTI) project [11]. ALGE was developed, using thermal imagery, to simulate power plant discharge waters into cooling lakes and other free surface water bodies (cooling canals, direct discharge to rivers, and oceans). Situations in which ice formation occurred were not considered in the earlier versions of the code. Originally, the inputs for ALGE, by definition, included physical parameters which describe the three-dimensional hydrodynamic and thermodynamic states of a body of water, the meteorological conditions at a given time, and the cooling pond bathymetry [13]. The outputs from the ALGE model included a surface and volume temperature distribution as well as a three-dimensional velocity flow map of the given body of water.

Recent work has been completed to validate the extension of the ALGE model to generate ice and snow when the water body is in a cold climate [12]. The ice and snow
extension added a snow thickness parameter as an additional input into the model. As a result of this extension, the ALGE model now also produces an ice thickness distribution. Figure 4.1 depicts simulated surface temperature maps for a pair of ALGE model runs for the same pond depicting the effects of varying weather conditions on the thermal distribution observed for the surface waters. A significant difference between total amount of ice coverage is evident with a larger portion of the pond covered by ice during the freeze-up period. While a larger area of the pond is ice-free during a warm-up period, there is still an almost discrete transition between the water and ice boundary layer.

The ALGE code optimizes the mass flow rate prediction for a given lake and weather conditions by incrementally changing the power generation facility’s operating parameters. The rate at which the thermal waste is discharged into the pond can be used in subsequent engineering process models to predict operating power levels for the power generation site [9]. A predicted thermal plume is produced for each combination of parameters and is compared to the observed thermal plume image. The parameters used to produce the thermal plume prediction that best matches the observed thermal plume are designated as the hypothesized plant operational parameters. The comparison process is depicted below in Figure 4.2.
4.2 Operating ALGE

In order to execute an ALGE simulation a series of input files, which adhere to strict FORTRAN formatting rules, need to be generated. Even though some of the information is spatial in nature (e.g. pond layout and bathymetry), these input files are limited to the form of text files. Each of the input files required are listed below with a description. Sample ALGE input files can be found in Appendix P.

- **param.dat**
  - This file contains all the simulation variables that are not temporally or spatially dependent. These variables include the number of nodes in the horizontal and vertical directions, the total simulation time, the temporal resolution, snow thickness, etc. The total list of variables is too long to list here and the reader is referenced to Appendix P for a complete listing.
• idepth.dat
  – This file specifies the depth of the body of water being simulated and is laid out as a grid of positive values. Each value represents the depth in units specified by the user. A depth of 0 means ground level, while any other positive value is the depth of the water column. The dimensions of the grid are dictated by the number of nodes in both the horizontal and vertical direction, as specified by the user in the param.dat file.

• igrid.dat
  – This file specifies the layout of the body of water being simulated and is organized as a grid of positive values. A value of 1 indicates the presence of water, a value of 0 indicates no water, a value of 7 indicates the heat discharge location, and a value of 6 indicates the location of the fluid intake. The dimensions of the grid are dictated by the number of nodes in both the horizontal and vertical direction, as specified by the user in the param.dat file.

• flow.dat
  – A file containing temporally varying flow rate data for the cooling pond. This is simply a text file with as many lines required to satisfy the time requirements of a simulation. Each line contains a single floating-point value which represents the flow rate of the heat discharge into the cooling pond at some desired time increment. The total number of lines is determined by dividing the total time of simulation required by the time increment.

• deltat.dat
  – A file containing the temporally varying temperature differential across the cool water intake and hot injection points. This is simply a text file with as many lines required to satisfy the time requirements of a simulation. Each line contains a single floating-point value which represents the temperature difference in degrees at some desired time increment. The total number of lines is determined by dividing the total time of simulation required by the time increment.
4.2. Operating ALGE

- **sfc.dat**
  - A file containing temporally varying surface meteorological conditions which match the temporal resolution of the simulation. This file is a series of columns of data. These parameters are total hours of simulation, wind direction, wind speed, air temperature, dew point, cloud fraction, cloud height, pressure, snow emissivity, date, and time.

- **dimar.inc**
  - This file contains references to parameters set in the param.dat file for FORTRAN array definition.

- **snd.dat**
  - This file contains sounding data for the upper air. These variables are temperature and perceptible water. ALGE expects the values in 12 hour intervals and uses a cubic spline to generate hourly data.

- **peramp.dat**
  - This file contains tidal forcing data (which has no variation for pond modeling).

- **seadens.dat**
  - This file contains containing water density values as a function of temperature and salinity.

- **srsfl2.dat**
  - If the modeled cooling pond has a secondary mass source (such as a river or stream), this file contains the flow data for this secondary source.

Once all of the input files are created and in one directory, the required FORTRAN files are compiled and executed from a terminal. Feedback from the ALGE routine regarding its progress can be viewed from the terminal or redirected to a log file as the process runs in the background. Upon completion of the model execution, several files will have been created containing the results. All results are stored in text files. The results contain information about the vertical temperature profiles through the columns of water, ice thickness and location, and the fluid velocity at a given position. Each of these output files contains the results data organized into arrays the size of the user-defined grid. The arrays are appended one after another with each increment of simulation time. In order
to extract an exact time, day, or depth, the user must know the total simulation time, the temporal resolution, and the grid size used to model the environment.

A suite of tools were created for the express purpose of interfacing with output files produced by the ALGE model. These tools are command line scripts written in IDL that allow a user to extract either the entire time series of resulting data or only a subset of specific points in simulation time. The resulting data is written out as standard TIFF image that can then be further processed. Additionally, tools were created for extracting pertinent information from the resulting imagery, such as the ice extent for a given point in time. All of these tools are documented in Appendix S.

### 4.3 RIT WASP system

RIT’s Wildfire Airborne Sensor Program (WASP) instrument, shown in Figure 4.3, is a multiple sensor aerial mapping system with broadband coverage in the infrared and visible spectrum. The sensor was built by the RIT Digital Imaging and Remote Sensing Laboratory (DIRS). WASP utilizes direct georeferencing hardware and processing techniques to create orthorectified imagery on-the-fly as the sensor is flown over the target scene.

![WASP sensor in laboratory.](image)

(a) WASP sensor head  
(b) WASP computer racks

Figure 4.3: WASP sensor in laboratory.
4.3. RIT WASP system

Originally designed as a wildfire detection and mapping system, WASP was built with three 640x512 pixel infrared cameras covering 0.9 -1.7\(\mu m\), 3-5\(\mu m\) and 8-9.2\(\mu m\); this spectral coverage allows the use of a multispectral technique for positively detecting the presence of a wildfire in the imaged scene. Each infrared (IR) camera has a 25 micron pixel pitch and a lens with an approximate focal length of 25mm. The system also carries a 4000x2672 pixel RGB camera with a 9 micron pixel pitch and 50mm lens [16]. Each camera’s optical system is stabilized for flight conditions and is geometrically modeled for lens distortion, principle point offsets, and focal length. To enable the creation of orthophotos by direct georeferencing, an Applanix POSAV-310 is utilized to record attitude information during the mission’s flight. The Applanix’s Litton LN-200 inertial measurement unit (IMU) is rigidly mounted to the camera frame assembly and boresight alignment angles for each camera are applied to generate exterior orientation parameters for each exposure station. Boresight angles for each camera are developed through a traditional bundle adjustment process utilizing highly-overlapped imagery, flown over a surveyed control point field or using a custom built calibration cage [16].

All imagery and metadata from the mission flight are recorded by a rack-mounted computer system on solid state removable media allowing for a service ceiling of at least 20,000 feet (tested) and high reliability without the use of specialized sealed hard drive enclosures. The WASP automated data processing computer (ADP) has the ability to orthorectify imagery on-the-fly as its collecting, utilizing real-time exterior orientation solutions calculated by the POSAV-310 and an archived digital elevation model of the area. Real-time generated orthophotos typically tie together acceptably for tactical applications and are absolutely accurate to about 4 meters RMS. The raw recorded data can be further refined in a post-processing workflow directly yielding georeferenced orthophotos absolutely accurate to better than 0.5 meters RMS [16].

4.3.1 WASP Sensor Blackbodies

The midwave infrared (MWIR) and longwave infrared (LWIR) array cameras on the WASP system are inherently susceptible to changes in environment that manifest as radiometric non-uniformities in the collected imagery. Most commercial IR camera systems provide mechanisms to perform uniformity corrections in a fixed environment; these procedures are not practical when applied to a constantly changing environment onboard an aircraft. As part of this work, two thermoelectric plate blackbody reference sources were added to
the system to address these issues. During a typical flight, the calibrators are moved to fill the field-of-view of the camera and imaged at two temperatures that bracket the expected temperatures in the scene to be mapped, typically at the beginning and end of each flight line. Imagery and temperatures gathered during the calibration process are used to post-process imagery from the MWIR and LWIR cameras to perform a non-uniformity correction and calibrate the images to sensor reaching radiance. The blackbody sources and their configuration on WASP are shown in Figures 4.4(a) and 4.4(b).

![Image of blackbody sources and WASP sensor head](image1)

Figure 4.4: Shown on the left in Figure 4.4(a) are the blackbody reference sources that are mounted on the WASP sensor head. Shown on the right in Figure 4.4(b) is an AutoCAD rendering of the WASP sensor head with the two blackbody reference sources mounted.

### 4.4 Ice coverage camera

The extent of ice coverage on the cooling pond was a significant parameter to observe. Due to weather restrictions on flight feasibility during winter months, a stationary camera system with a fisheye lens was constructed and mounted on the roof of the main power plant facility building. A Sigma fisheye lens was mounted onto a Nikon D50 digital SLR and used to capture a time series of the cooling pond from this high vantage point. This camera system was capable of producing a 180° FOV image of the cooling pond in a single exposure. The methodology used to extract the ice extent from the resulting distorted images is described in Section 5.1.3.
4.5 Data validation site

The Midland Cogeneration Venture (MCV) in Midland, MI was chosen as the validation site for this research. MCV is a gas-fired power and steam cogeneration facility. The plant operates with a baseline power load of 200 MW to supply steam to a nearby industrial facility and has the ability to spike to 1500 MW based on grid demands. The facility uses both cooling towers and a cooling pond to serve its cooling requirements. The waste heat injected into the cooling pond ranges from 200 to 500 MW. The cooling pond is a man-made water body and covers an area of 3.7 km$^2$ (880 acres). The average depth of the pond is approximately 30 feet, with a large reserve reservoir near the cold water intake at a depth of 60 ft. Figure 4.5 is an aerial image of the MCV facility.

![Aerial view of MCV facility and cooling pond. Power facility is located on the northern end of the pond with a concrete berm separating the hot out flow on the left from the cold intake on the right.](image)

This particular site satisfied a unique set of conditions that proved to create an ideal area of study. In order to collect the appropriate validation data for the ALGE ice formation modification, the site was required to freeze during the winter. Typically, industrial cooling ponds are designed to be immune to freezing in an effort of maximize the cooling efficiency of the pond. When ice forms on the surface of the pond it leads to conductively-driven cooling, a significantly slower cooling process than convective heat loss. However, MCV was originally intended as a nuclear power plant and would have required a large
cooling pond to be able to effectively cool the proposed amount of rejected waste heat in the pond. Due to lack of funds, regulatory issues, and construction problems, the plant was converted to a gas-fired plant after 85% of the original construction had been completed - including the construction of the cooling pond. Because the power load generated by the gas-fired plant is significantly lower than the projected power load of the nuclear plant, the waste heat entering the pond is not significant enough to maintain an ice-free body of water during the winter months.

4.6 Summary

This chapter reviewed the three different systems that will be merged together to execute the implemented approach aimed at improving the modeling of a cold climate environment. These systems included the imagery collection platform, hydrodynamic modeling code, and the ice coverage camera system. Additionally, the ground site chosen for model validation was identified. The next chapter will discuss the methodology which implemented all these systems to complete this research.
Chapter 5

Methodology

The following chapter will walk through the data collection campaigns and the steps taken to build the PSO-ALGE architecture. Each of the systems used for the ground truth campaign are outlined as well as their exploitation. Extensive work was performed to investigate how to appropriately evaluate an ALGE simulation and compare simulation results to actual observations. The PSO architecture and how ALGE was implemented to run within the RIT Research Computing (See Appendix Q.2.

5.1 Data collection

A large scale, multi-year data collection campaign was executed in order to accomplish the validation of the ALGE ice and snow extension. These campaigns yielded a database of seven ground-truthed, calibrated, thermal datasets of the power plant facility in Midland, MI. These aerial data sets provided some of the empirical data to be used for the optimization’s functional evaluation. Data campaigns were carried out across two winters from November of 2008 to April of 2010. The collected ground-based data was used to calibrate the acquired thermal imagery. When weather became a limiting factor in flying the WASP sensor, digital aerial images of the pond were acquired using a Canon DSLR camera by a pilot in the Midland, MI area. The stationary camera system that was mounted on the roof of the main power plant facility building continuously monitored the ice extent across the pond. These ancillary images proved indispensable for fortifying the ice coverage data set. Additionally, sets of buoys were designed, built, deployed, determined to have failed, re-designed, and deployed again over the course of the two winters. While the original
purpose of the buoys was to continually collect data on the thermal conditions in the cooling pond, the results from these systems collected over both winters were not used in the final workflow. However, the experience in managing these assets led to many insights into quality measurement taking in extreme weather conditions. Appendices I.1 and I.2 describe these systems and their modifications.

5.1.1 Autonomous weather station

A single weather station was located near the hot water injection point at the power plant. The weather station was constructed entirely from Campbell Scientific products and remained unchanged throughout both winters except for the addition of a second solar panel during the 2009-2010 winter. A Campbell Scientific CR3000 datalogger queried all individual weather modules at 30-second intervals and recorded averages or instantaneous values, depending on the module, at five-minute intervals. All recorded data was stored into data files that were transmitted twice daily (noon and midnight) via a cellular modem imbedded in the system. Data recorded by the weather station included air temperature, relative humidity, precipitation amount, wind speed, wind direction, shortwave irradiance, longwave irradiance, and barometric pressure. The constructed weather station is shown in Figure 5.1.

Figure 5.1: Weather station constructed on the northern shore of the MCV cooling pond
5.1. Data collection

5.1.2 Manual ground truth collection

Due to the strong dependence of ground truth collection on favorable weather conditions, a RIT ground team was stationed in Midland, MI at all times for the duration of the 2008-2009 experiment period. During the first winter all ground truth measurements were made from an airboat in order to minimize the chance of any personnel injury on thin ice regions of the lake. The airboat offered the unique ability to not only maneuver on water but also on ice. In lieu of making ice and snow measurements from the boat, transects of data were also collected by walking onto the ice from the shore of the cooling pond.

![Figure 5.2: Photos from ground truth campaign. Figure 5.2(a) shows a team member measuring water temperatures with a thermistor. Figure 5.2(b) shows the airboat used for data collection.](image-url)

During the second winter data campaign, the RIT ground team traveled between RIT and Midland when weather permitted. Due to more reliable buoy instrumentation, there was little interaction required by the team to maintain the equipment in working order. The team worked from a pontoon boat for the entirety of the winter and relied on buoy-derived ice thickness estimations.

Data collected by the ground truth team included localized relative humidity, water surface temperature, and bulk water temperature. All ground truth measurements were made from a boat and were recorded immediately following the collection of aerial imagery.
It was not advisable to collect surface temperatures concurrently with the flight due to the thermal influence the boat would have on the surface temperatures of the exposed water. It was assumed there was approximately a one-hour window following the imagery collection where any collected surface temperatures were valid. Bulk temperature measurements were made with both a contact thermistor mounted on a styrofoam float and an Omega HH41 temperature probe. Surface temperatures were observed using both an Omega OS36 infrared radiometer and a Heitronics KT19.82 infrared radiometer. Positioning information was recorded using a handheld Garmin E-Trax GPS and localized weather data (i.e. relative humidity and wind speed) was collected using a handheld Kestrel 4000 weather meter.

5.1.3 Ice cover estimation

To facilitate the collection of ice extent data, irregardless of flight conditions, a camera system was designed and installed on the roof of the main power plant facility building. This system incorporated the use of a fisheye lens to capture a 180° FOV image of the cooling pond in a single exposure. The resulting images had geometrical distortions introduced by the lens. To acquire the necessary parameters to remove the geometric lens distortion, a calibration cage maintained by the DIRS research group at RIT was imaged. A block bundle adjustment was performed to solve for focal length, symmetric radial distortion, and de-centering distortion. The results of this adjustment were applied to the rooftop images using a program written in IDL. Each pixel was transformed onto a new grid based on the distortion coefficients. The image was then interpolated back onto a regular grid using a radial basis function [4]. An example of this process is shown in Figure 5.3. The pixels in the distortion free image that contain ice or water were identified by hand using the ROI tool within ENVI as shown by Figure 5.4.

In order to determine the area of each pixel from the undistorted image, the location of the horizon needed to be found in each image. The distance from the center of the image to the horizon was found using edge detection. A region of interest (ROI) is drawn which contains the horizon. Within this region a “rake” of vertical lines are created. The points which contain the strongest falling edge (from bright sky to dark horizon) were found and then used to find the horizontal best fit line as well as the angle of the line relative to the x-axis. This process was implemented in the Labview environment. The geometry of the image and its content are shown in Figure 5.5(a).
5.1. Data collection

(a) Original from roof-top. (b) After de-fish process

Figure 5.3: Original image captured from roof-top with processed version with geometrical distortions from fisheye lens removed. It should be noted that pixels near the edges of image begin to exhibit correction errors. This is due to low point density of calibration grid at the edge of the FOV.

Figure 5.4: Ice and water pixels have been identified with two different ROI's. Water is tagged as blue, ice as red.

To convert each pixel into a unit of area three things needed to be known: the height of the camera relative to the pond surface ($H'$), the focal length of the lens ($f$), and the true depression angle ($\theta$). The true depression angle was measured from the camera’s optical axis to the true horizon line. The geometry is shown in Figure 5.5(b).

$H'$ was found by comparing the height of the building to the pond surface using a hand-held GPS unit. The focal length was an output of the lens correction routine. The true depression angle, $\theta$, was calculated using equations and a process described by Wolf [27]. Once $H'$ and $\theta$ were found, these parameters were used to rotate and translate the image so that the center/origin of the image aligned with true horizon and was parallel to the x-axis. For each pixel the lengths along the vertical ($\Delta y$) and horizontal ($\Delta x$) sides...
Figure 5.5: Tangent line needed for calculations of scale shown in 5.5(a). Length OK′ and the angle of the tangent line relative to the X-axis are needed. Side view of principle plane of the oblique photograph shown in 5.5(b).

were calculated in pixel units from two of the image’s corners. Both lengths (Δx, Δy) were projected into the world plane and translated into arbitrary world coordinates (see Equations 5.2 and 5.3). It should be noted that the pixel shape was distorted from a square into a polygon. However, since the camera height was small relative to most airborne scenarios, the error introduced into the area calculation was assumed to be negligible. The area was now found by Equation 5.1.

\[
A = \Delta X \cdot \Delta Y
\]  

(5.1)
5.1. Data collection

\[ \Delta X = \Delta x \left( \frac{H'}{|y_h| \cos \theta} \right) \]  
\[ \Delta Y = \left( \frac{1}{y_l} - \frac{1}{y_h} \right) \left( \frac{fH'}{\cos^2 \theta} \right) \] 

In order to determine the accuracy of this implemented approach, two data sets representing the best and worst case scenarios were compared. The results extracted using the oblique images were compared to ice cover estimation values calculated from overhead nadir imagery captured by the WASP system on the same days and are show in Table 5.2. Images collected on 24 February 2009 show most of the pond uniformly covered in ice and therefore allows the user to create accurate ROIs in both the nadir and oblique images; this data set represents the “best case scenario”. Imagery collected on 4 March 2009 illustrates a variable ice distribution on the lake surface. This type of ice distribution is difficult to distinguish in the oblique imagery as the majority of the exposed water is obscured by the perspective and represents a “worst case scenario”. Figure 5.6 shows each day used in the comparison as both an oblique image, a nadir LWIR image, and a nadir LWIR image with the open water ROI selected.

<table>
<thead>
<tr>
<th>Date</th>
<th>Image Type</th>
<th>Total Area [acres]</th>
<th>Water Area [%]</th>
<th>Ice Area [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>02/24/09</td>
<td>Oblique</td>
<td>853.570</td>
<td>7.62</td>
<td>92.34</td>
</tr>
<tr>
<td></td>
<td>Nadir</td>
<td>853.810</td>
<td>9.28</td>
<td>90.71</td>
</tr>
<tr>
<td></td>
<td><strong>Difference</strong></td>
<td><strong>0.02%</strong></td>
<td><strong>1.76</strong></td>
<td><strong>1.63</strong></td>
</tr>
<tr>
<td>03/04/09</td>
<td>Oblique</td>
<td>853.722</td>
<td>52.01</td>
<td>47.99</td>
</tr>
<tr>
<td></td>
<td>Nadir</td>
<td>854.182</td>
<td>43.71</td>
<td>56.29</td>
</tr>
<tr>
<td></td>
<td><strong>Difference</strong></td>
<td><strong>0.05%</strong></td>
<td><strong>8.3</strong></td>
<td><strong>8.3</strong></td>
</tr>
</tbody>
</table>

Table 5.1: Pond Area Comparison (oblique vs. nadir)

Indicated by the results in Table 5.2, the approach performed well. For the ideal conditions occurring on the 24th of February, both the open water and ice coverage estimations derived from the oblique imagery deviated from the same estimations derived from nadir imagery by 1.76% and 1.63%, respectively. Calculations performed on data from the “worse case scenario” day, the 4th of March, still only show a percentage difference of 8.3% for both calculations. Considering the approach used, the small errors in estimation were considered more than acceptable and this technique was used to collect ice cover estimations.
CHAPTER 5. Methodology

(a) 2/24/09 - Oblique  (b) 2/24/09 - Nadir LWIR  (c) 2/24/09 - Nadir LWIR with open water ROI
(d) 3/04/09 - Oblique  (e) 3/04/09 - Nadir LWIR  (f) 3/04/09 - Nadir LWIR with open water ROI

Figure 5.6: Images used in comparison and error metric calculations. Left-most image shows de-fished fisheye images. Middle images show nadir captured LWIR images used to select the open water ROIs for error comparison.

5.1.4 Calibration of WASP imagery

An ad-hoc calibration technique was chosen for the thermal calibration of WASP flights over the MCV site. A review of this type of method is in Appendix B.2. A detailed description of the how this technique was implemented for this application is in Appendix E. It should be noted that this technique could have been applied in a way as to skip the conversion of digital counts to radiance space. The implication of that choice being there would be a direct relationship established to measured ground temperature and the digital count observed at the sensor. The removal of a radiance conversion from the process would reduce the amount of calculated error in the calculated temperatures. However, in the interest of implementing a sensor-based radiance calibration that would increase the abilities of the WASP sensor to perform other data acquisition campaigns, a process was put in place to produced calibrated sensor-reaching radiance images. The code written to
accomplish the calibration can be found in Appendix R.

5.2 Evaluation of ALGE simulations

To determine the validity of an ALGE calculated simulation, the surface temperature distributions and ice coverage estimations generated by ALGE needed to be compared against calibrated LWIR data collected by the WASP sensor as well as ground and aerial-based observations of ice coverage (categorized as image-derived information). In order to fully implement this comparison, a metric was developed to determine the quality of a given simulation output when compared to the image-derived information.

In previous optimization approaches, the RMS (root mean squared) error was calculated using every point in the water body in both the simulated and actual imagery [14]. This metric produced a single RMS value for a given comparison between a simulation and an actual image. However, the current version of ALGE has been extended to be operational in cold-climate conditions. As a result, ice and snow can occupy portions of a simulated body of water. The introduction of ice and snow into the scene removes the possibility of an image-to-image direct comparison because of the emissivity differences introduced by the ice and snow. In order to derive a comparison metric, a new, modified RMS solution was proposed and compared to a standard RMS solution. Both approaches are applied to thermal and ice coverage data to evaluate the model’s ability to produce valid surface temperature distribution estimations and valid ice coverage estimations. For each calibrated thermal image a collection of geo-referenced temperature points are collected from each acquired thermal image to serve as the collection of observed thermal measurements. Ice coverage values were derived from all possible imagery sources and served as the collection of observed ice coverage values.

5.2.1 Root Mean Squared Error (RMSE)

The equation to calculate the RMS error for a set of observed ($O_i$) and expected ($E_i$) values is shown below in Equation 5.4 where $N$ is the total number of samples. A RMS error value can also be normalized, $RMS_{norm}$, by dividing the quantity by the range of observed values, shown in Equation 5.5. The resulting value will range from 0 to 1 and represent the amount of residual variance in the data set. The lower the normalized RMS value, the less residual variance in the set and the better the simulated values match the
observed values.

\[
RMS = \sqrt{\frac{\sum_{i=0}^{N} (E_i - O_i)^2}{N}} \quad (5.4)
\]

\[
RMS_{\text{norm}} = \frac{RMS}{O_{\text{max}} - O_{\text{min}}} \quad (5.5)
\]

While a normalized RMS error value will indicate the level of relative variance that a given simulation has as compared to other simulations, the value itself is susceptible to outliers in the observed population. If an extreme value exists on either end of the observed range, the denominator in the \(RMS_{\text{norm}}\) calculation becomes large and reduces the range distribution of calculated normalized RMS values. Also, since there is no restriction on the data sets using this metric, it is possible the denominator could be calculated as zero.

### 5.2.2 Modified Root Mean Squared Error (M-RMSE)

Similar to a standard RMS error calculation, the described quantity uses the difference between the expected and observed values of a model to determine how well the given model describes the observed behavior of a system. This technique was developed and tested specifically for this work under the motivation to avoid a situation where the quantity would have a denominator of 0. The new metric is shown in Equation 5.6 where \(E_i\) (Equation 5.7) and \(O_i\) (Equation 5.8) represent the de-meaned, N-element time series for both the expected and observed values, respectively.

\[
RMS_{\text{mod}} = \sqrt{\frac{\sum_{i=0}^{N} \left( \bar{E}_i^2 - 2\bar{E}_i\bar{O}_i + \bar{O}_i^2 \right)}{\sum_{i=0}^{N} (\bar{E}_i^2 + \bar{O}_i^2)}} \quad (5.6)
\]

\[
\bar{E}_i = E_i - \frac{\sum E_i}{N} \quad (5.7)
\]

\[
\bar{O}_i = O_i - \frac{\sum O_i}{N} \quad (5.8)
\]

The modifications to the standard RMS calculation to produce Equation 5.6 bounds the possible values for \(RMS_{\text{mod}}\) between -1 and \(\sqrt{2}\). In addition, \(RMS_{\text{mod}}\) can never have a denominator equal to 0, unless the time series being examined are both zero at all
elements. It is important to note that the middle term in the numerator of Equation 5.6 represents the correlation coefficient for the two sets. If the observed and expected data is uncorrelated, the numerator reduces to the denominator and produces a metric value of 1. If the two data sets are perfectly correlated then the entire quantity reduces to 0.

5.2.3 Application to data sets

Data was collected over two winters and has produced 7 quality, calibrated thermal images of the Midland Cogeneration Venture (MCV) cooling pond. In addition there are 14 days during the first winter and 25 days during the second winter where the percentage of the total ice coverage was observed either from aerial imagery or from stationary imagery collected using a roof-mounted camera. Because the goal of the ALGE simulation is to accurately model the conditions of a given environment throughout the entire winter, a given simulation will be compared to all data collected within a winter. For example, if the ALGE model is run for the 2008-2009 winter season, water comparisons will be made for each day simulated that correspond to a day for which there exists observation data (i.e. the 3 days worth of calibrated thermal imagery collected during that winter). In addition, the modeled ice coverage will be compared at 14 simulation days that correspond to the dates of collection for the observed ice coverage data.

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Winter 2008-2009</th>
<th>Winter 2009-2010</th>
</tr>
</thead>
<tbody>
<tr>
<td>Imagery</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16 February 2009</td>
<td>12 February 2010 (day)</td>
<td></td>
</tr>
<tr>
<td>24 February 2009</td>
<td>12 February 2010 (night)</td>
<td></td>
</tr>
<tr>
<td>4 March 2009</td>
<td>4 March 2010 (day)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4 March 2010 (night)</td>
<td></td>
</tr>
<tr>
<td>Ice Fraction</td>
<td>14 days</td>
<td>25 days</td>
</tr>
<tr>
<td>Temperature Pts.</td>
<td>2900 over 4 days</td>
<td>1324 over 4 days</td>
</tr>
</tbody>
</table>

Table 5.2: Overview of data sets for both winter collection campaigns

Two different calculations were performed for each modeled winter: one for the water temperature distributions and one for the ice coverages. These calculations were repeated using both the standard RMSE and the modified-RMSE metrics. The two parameters that were used to determine the validity of a given simulation are as follows:

1. the ALGE-modeled temperature distribution at the water surface compared to the observed temperature distribution of the pond by the LWIR sensor, and
2. the ALGE-modeled percentage of total lake area that is covered with ice compared to the observed percentage of ice coverage observed by the WASP sensor, ground-based observations, and aerial handheld observations.

5.2.4 Metric calculation for water temperature

For a given simulation, the observed data set, $O_{water}$, was a set of temperatures for each comparison day and is defined in Equation 5.9, where $K$ was the total number of points in the set. Each set was comprised of temperatures chosen based on pre-determined geographical coordinates. Similarly, the expected data set, $E_{water}$ (defined in Equation 5.10), was a set of temperatures extracted from the simulated thermal imagery at the same geographical locations on the same day as the corresponding thermal imagery was acquired. The set of pre-determined coordinates were generated using tools within ENVI to extract point information from the thermal imagery. The set for the first winter contained a total of 2900 points collected from all 3 LWIR images. The set for the second winter contained a total of 1324 points collected from all 4 LWIR images. Each set of points was chosen to exclude any locations where ice is observed in the WASP imagery. Figure 5.7 shows an example data set with the selected temperature points.

Figure 5.7: WASP image collected 10 March 2010 with collected temperature points overlaid.
5.2. Evaluation of ALGE simulations

\[ O_{\text{water}} = [T_{\text{obs},1}, T_{\text{obs},2}, T_{\text{obs},3}, \ldots, T_{\text{obs},K}] \]  
(5.9)

\[ E_{\text{water}} = [T_{\text{est},1}, T_{\text{est},2}, T_{\text{est},3}, \ldots, T_{\text{est},K}] \]  
(5.10)

With these data set definitions, the de-meaned versions were calculated as follows (Equations 5.11 and 5.12).

\[ O'_{\text{water}} = O_{\text{water}} - \frac{\sum_{k=0}^{K} O_{\text{water},k}}{K} \]  
(5.11)

\[ E'_{\text{water}} = E_{\text{water}} - \frac{\sum_{k=0}^{K} E_{\text{water},k}}{K} \]  
(5.12)

The modified-RMS metric, \( RMS_{\text{mod,water}} \) is calculated was shown in Equation 5.13.

\[ RMS_{\text{mod,water}} = \sqrt{\frac{\sum_{k=0}^{K} \left( E'_{\text{water},k}^2 - 2E'_{\text{water},k}O'_{\text{water},k} + O'_{\text{water},k}^2 \right)}{\sum_{k=0}^{K} \left( E'_{\text{water},k}^2 + O'_{\text{water},k}^2 \right)}} \]  
(5.13)

The standard RMS metric, \( RMS_{\text{norm,water}} \) was calculated as shown in Equation 5.15. The standard RMS calculation did not use the de-meaned version of the expected and observed datasets.

\[ RMS_{\text{water}} = \sqrt{\frac{\sum_{k=0}^{K} (E_{\text{water},k} - O_{\text{water},k})^2}{K}} \]  
(5.14)

\[ RMS_{\text{norm,water}} = \frac{RMS_{\text{water}}}{O_{\text{water,max}} - O_{\text{water,min}}} \]  
(5.15)

5.2.5 Metric calculation for ice coverage

To apply the metric to ice coverage, the observed data set, \( O_{\text{ice}} \), was a the set of ice coverage percentages observed either from handheld aerial, mounted aerial, or roof camera imagery throughout the winter. The expected data set, \( E_{\text{ice}} \), was the modeled ice coverage percentages from the ALGE simulation that correspond to the observation days. \( J \) represented the total number of point sets in the population.
\[ O_{\text{ice}} = [P_{\text{obs},1}, P_{\text{obs},2}, P_{\text{obs},3}, \ldots, P_{\text{obs},J}] \] (5.16)

\[ E_{\text{ice}} = [P_{\text{est},1}, P_{\text{est},2}, P_{\text{est},3}, \ldots, P_{\text{est},J}] \] (5.17)

With these data set definitions, the de-meaned versions are calculated as follows (Equations 5.18 and 5.19).

\[ O'_{\text{ice}} = O_{\text{ice}} - \frac{\sum_{j=0}^{J} O_{\text{ice},j}}{J} \] (5.18)

\[ E'_{\text{ice}} = E_{\text{ice}} - \frac{\sum_{j=0}^{J} E_{\text{ice},j}}{J} \] (5.19)

The modified RMS metric, \( RMS_{\text{mod,ice}} \) is calculated as shown in Equation 5.20.

\[ RMS_{\text{mod,ice}} = \sqrt{\frac{\sum_{i=0}^{J} \left( E'_{\text{ice},j}^2 - 2E'_{\text{ice},j}O'_{\text{ice},j} + O'_{\text{ice},j}^2 \right)}{\sum_{j=0}^{J} \left( E'_{\text{ice},j}^2 + O'_{\text{ice},j}^2 \right)}} \] (5.20)

The standard RMS metric, \( RMS_{\text{norm,ice}} \) is calculated as shown in Equation 5.22. The standard RMS calculation does not use the de-meaned version of the expected and observed datasets.

\[ RMS_{\text{ice}} = \sqrt{\frac{\sum_{j=0}^{J} (E_{\text{ice},j} - O_{\text{ice},j})^2}{J}} \] (5.21)

\[ RMS_{\text{norm,ice}} = \frac{RMS_{\text{ice}}}{O_{\text{ice,max}} - O_{\text{ice,min}}} \] (5.22)

### 5.2.6 RMS Metric comparison

The impact of both the observed parameters (temperature conditions and ice coverages) on the overall thermodynamic environment was investigated in order to evaluate both metrics’ responses to changes in the parameters. A simulation was completed using known plant operational data, observed meteorological data, and a simple snow coverage model (10 cm blanket) for the 2008-2009 winter. The resulting simulated ice fractions demonstrated a
reasonable correlation with measured ice fractions as shown in Figure 5.8.

![Figure 5.8: Comparison of simulated and measured ice fractions using measured environmental and plant data from the 2008-2009 winter.](image)

Because of this demonstrated correlation, confidence was given to the simulated data, allowing it to be used for further investigation of metric behavior. The waste heat load from the plant was varied for additional simulations to determine the relationship between simulated ice fraction and the heat load being injected into the cooling lake. The heat load, $Q$, was varied at levels of 0%, 50%, 150%, 200%, 250%, and 300% of the measured load. Both RMS metrics were calculated for the ice fraction and surface temperature data resulting from the different simulations. Figures 5.9 and 5.10 show the results from the 2008-2009 and 2009-2010 season simulation data, respectively.
Figure 5.9: Metric sensitivity to 2008-2009 data set. $RMS_{\text{mod,ice}}$ and $RMS_{\text{mod,water}}$ represent the metric data using the modified RMS method while $RMS_{\text{norm,ice}}$ and $RMS_{\text{norm,water}}$ represent the metric data using the traditional RMS calculation.
5.2. Evaluation of ALGE simulations

(a) Water comparison

\( \text{RMS}_{\text{mod,water}} \) and \( \text{RMS}_{\text{norm,water}} \) represent the metric data using the modified RMS method while \( \text{RMS}_{\text{norm,ice}} \) and \( \text{RMS}_{\text{mod,ice}} \) represent the metric data using the traditional RMS calculation.

Figure 5.10: Metric sensitivity to 2009-2010 data set.

(b) Ice comparison
An ideal metric response to each data set would show the lowest metric value occurring at the 100% waste heat load level and produce increasingly higher values as the waste heat load is increased from this baseline value. Additionally, it would be expected that the simulation at 50% waste heat load would produce a worse simulation than the 100% level simulation and thus would have a higher metric value. From the results displayed in Figures 5.9 and 5.10, one can see this behavior is observed to an extent, however, neither metric performs ideally. In both seasons, however, the traditional RMS metric was more aligned with the expected result.

Based on the observed metric results for simulations completed at different waste heat loads, the traditional RMS metric performed slightly better than the modified RMS metric. As a result, this metric approach was implemented in evaluating a given simulation during the optimization process.

### 5.2.7 Metric combination

The two resulting metric values, for ice fraction and surface temperature, separately quantify how the model compared to both the observed temperature conditions and ice coverages. In order to implement this functional evaluation into an optimization routine, either these two values need to be combined into a single value or only one value needs to be chosen to be representative of the accuracy of a simulation. A simple linear combination is proposed in the event both metrics are required to describe the accuracy of a simulation. This linear combination is shown in Equation 5.23, where $\alpha$ and $1 - \alpha$ represent weighting factors.

$$R_{\text{total}} = (\alpha)RMS_{\text{water}} + (1 - \alpha)RMS_{\text{ice}}$$  \hspace{1cm} (5.23)

### 5.2.8 Metric weighting

The average ice coverage for each simulation was calculated and compared to the corresponding average waste heat load for the simulation duration. This comparison is shown in Figure 5.11. Additionally, for each simulation performed at different waste heat levels, the average water temperature was compared to the average air temperature via a differencing function. This temperature difference was compared to the average waste heat load simulated in the cooling pond. This comparison is shown in Figure 5.12.
5.2. Evaluation of ALGE simulations

Figure 5.11: Comparison of average simulated ice fraction to waste heat load. Each point represents the average simulated ice fraction for an entire winter simulation.

Figure 5.12: Comparison of average difference between average water temperature and air temperature to waste heat load. Each point represents the average difference between the two temperature values for an entire winter simulation.
CHAPTER 5. Methodology

The observed relationships between the average waste heat load into the body of water and the two parameters are both nearly linear, while exhibiting an almost quadratic behavior at the tail. However, the relationships are inversely related to one another. From a physical standpoint, this inverse correlation is expected. As the average temperature of the water being injected into the pond increases with an increased waste heat load, the ice would recede further from the injection point in the pond, producing a positive correlation between waste heat and temperature and a negative correlation between waste heat and ice fraction.

As a result, it can be concluded that insight into both the thermal distribution and ice coverage parameters for a particular simulation are of equal importance when evaluating a simulation. The same level of confidence can be drawn from a simulation comparison performed by examining either the temperature distribution or the ice fractional coverage, or a straight combination of them where both parameters are equally weighted.

5.3 Correlation between ice extent and heat load

While the implementation of a PSO-driven ALGE model represents the majority of the work in this document, it is important to remember that the main ambition of the ALGE model is to model the thermodynamic conditions of a cooling pond to an acceptable level of accuracy in cold climate conditions. The confidence in this accuracy and the resulting ALGE outputs ultimately yield the information necessary to infer the current working conditions of a power plant facility. From this perspective the insight gained and described in Section 5.2.8, as well as work presented in Garrett et al. [12], are significant.

Upon completion and validation of the cold climate extension to ALGE, Garrett et al.[12] investigated the relationship between the average heat load injected into a cooling pond and the resulting average ice coverage over the simulated winter time periods. The data collected at the Midland site was used to execute 5 simulations where meteorological conditions were held constant. The heat load, $Q$, that the model injected into the cooling pond was varied from 0%, 50%, 100%, 200%, and 300% and generated synthetic data for the ice coverage at these different conditions. Figure 5.13 shows the temporal variation in ice coverage for the duration of simulation time for each heat load condition.

As indicated in Figure 5.13, when the heat load is brought to 0% the cooling pond maintains a constant ice coverage for the majority of the winter as expected. Conversely,
when the heat load was set to 300% the cooling pond stayed relatively ice free, with the exception of a very cold period of time during the middle of winter. The relationship between the average heat load and the average ice fractional coverage is shown in Figure 5.14.

The nearly linear connection demonstrated between the heat load and ice coverage further indicates that the ice coverage is generally a robust indicator of the average power being generated by the power facility for the duration of the winter. Garrett et al. [12] go on to attribute this simple correlation to the negative feedback loop that exists between the heat transfer from the lake to the atmosphere and the ice cover, despite the highly non-linear 3-D thermodynamic and hydrodynamic relationship used to describe the melting and freezing. In combination with the Section 5.2.8 results, these results support the position that the ice fraction is a valid observable for model performance validation due to the linear relationship between that and heat load.

It is because of this established linearity that the ice fractional coverage is used as the sole driving validation parameter in the PSO-ALGE simulations. This conclusion has interesting repercussions for the practical application of this work. Because the ability of a simulation to accurately model a given environment can equally be evaluated by either comparing thermal data or observed ice fractional coverage, the presence of quality thermal image data is not necessarily required to assess the validity of a simulation. Therefore, the model may be evaluated when the only data available is visible imagery. It is important to
CHAPTER 5. Methodology

Ice Cover vs Q Sensitivity

Figure 10. Average simulated ice cover over MCV cooling lake as a function of Q.

The 5 instrumented buoys RIT deployed at different locations on the Midland cooling lake measured profiles of water and ice temperature that can be compared to ALGE simulations. Since these are point measurements (unlike the ice coverage, which is an integral quantity), the degree to which these data are representative of the part of the lake that they were located in is unknown. RIT found that the automated method for measuring ice thickness at Shiva and Surya stations often differed from hand measurements by 30 to 40%. The Shiva station was located fairly close to the cooling water intake, which is the coldest and iciest part of the lake. Figure 11 compares the time series of measured ice thicknesses at Shiva to the results of 4 simulations which used different combinations of 3 or 10 cm snow depth and empirical or one dimensional longwave radiation transfer (1D LWIR) models. The simulations that used 3 cm of snow produced the best agreement with measurements earlier in the time series but did worse than the simulations that used 10 cm of snow late in the simulations. Based on these results, it appears that simulated ice thickness is more sensitive to snow cover than the ice area, although the greater uncertainty of the automated ice thickness measurements and the fact that these are point measurements (rather than area-integrated) makes this conclusion more tentative. Unlike the ice cover results, the simulated ice thicknesses were not sensitive to the type of downwelling thermal radiation model. Figure 11 shows that there was a sudden decrease in measured and simulated ice thickness at Shiva around February 9. This event was the result of a sudden increase in air temperature from below freezing to about 5°C that was accompanied by strong winds. The strong winds increase the rate of transfer heat from the air to the ice surface, thus speeding the rate of melting. If the wind direction is from the warmer part of the lake to the colder (ice-covered) part of the lake, then a second mechanism acts to increase the rate of melting. In this situation, the winds increase the speed of the current that carries above freezing water underneath the ice. During the period from February 7 to 10, above-freezing air temperatures combined with strong winds to cause melting from above and below the ice layer. As a result, the measured ice thickness dropped to zero, and all 4 simulations predicted much thinner or no ice.

Figure 5.14: Comparison of average heat load and average ice fractional coverage for each heat load condition.

note however, that this assumption assumes some knowledge of the temperature differential between the intake and output point of the cooling pond. For these experiments, this information was measured and known. For the proposed application to work in absence of thermal data, the temperature differential would have to be an optimized parameter.

5.4 Application of PSO to ALGE

In a traditional ALGE implementation, a user sets up a single simulation using input files, executes the code, and allows the simulation to run. The results produced from the simulation are compared to validation data derived from an image source (or sources). Depending on how well the model performed, the original input parameters are either accepted or changed. In an effort to create a more systematic and efficient approach in validating an ALGE simulation, PSO was applied to the parameter selection for the ALGE model. ALGE simulations were instantiated to run for simulation times representing entire winters. Inside the particle swarm optimization (PSO) paradigm, some input parameters for an ALGE simulation were generated via the evolutionary optimization process and
some were pulled from constant parameter sets. Simulations were evaluated using real data that, due to the environmental challenges of cold weather data collection, was collected at sparse and irregular intervals.

5.4.1 Optimizing parameter choice

In order to determine which ALGE input parameters contained the uncertainty that needed to be reduced via PSO optimization, all inputs were examined. The possibilities included the meteorological conditions, the plant operation parameters, or both. There are challenges and limitations to all three choices, as well as different ways of approaching how to optimize the parameters.

As meteorological conditions are temporal in nature, they can be extremely variable and difficult to model. If a parameterized function exists which can model these conditions, then the variables defining this function become the parameters of each particle. The bounding values for these parameters are generated by the physical limitations on the environment. If there is not a functional basis that describes the meteorological variation, a scaling parameter has to be used to increase or decrease an entire time series of data by a defined multiplier. For example, one could say they want to vary the air temperature and wind speed within a range \( \pm 15\% \) from a baseline approximated time series of those variables. The parameters to be optimized in the PSO scheme would be the two scalar weights for both weather variables. The solution space would be bound by \( \pm 15\% \).

The plant parameter inputs to the ALGE model are also temporal. These inputs include the temperature differential across the injection and intake sites in the pond, as well as the flow rate of the effluent used to dissipate heat into the cooling pond. When examining an entire winter, at hourly simulation intervals, these inputs become approximately a 2,600-element time series with each value representing either the flow rate or temperature differential at a given point in simulation time (24 hours/day x 108 days/winter). Similar to the meteorological conditions, if these inputs can be modeled using some type of parameter-driven function, the number of these parameters become the dimensionality of the solution space. However if these variables are inherently related to one another, then modeling them accurately requires this correlation to be accounted for, resulting in an arduous task.
5.4.2 Temporal input averaging

In an already complex modeling environment, the PSO engine was only allowed to control a single parameter for a given simulation, the temporal flow rate for the modeled cooling pond’s heated effluent. For clarity, it is important to highlight that while the flow rate is in fact a single parameter from the perspective of the ALGE model, it is not a single-dimensional parameter from the PSO perspective. For ALGE the flow rate is an array of values representing the time series of flow rates for the desired amount of simulation time. For example if a simulation was created to run for 300 hours of simulation time the input expected by the ALGE model would be a text file containing 300 values, each one representing the flow rate per hour. Thus, the flow rate is a single input parameter for ALGE. Inside the PSO paradigm, the flow rate is a multi-dimensional parameter that defines a particle. The flow rate cannot be represented by a single value; it is a temporal series of data. Given the example mentioned previously, if nothing is done to either parameterize or distill the flow rate, each particle in the swarm would be defined by 300 parameters (or flow rate values), one for each hour of simulation time. Because a particle’s dimensionality is analogous to the dimensionality of the solution space, it is clear that a reduction in dimensionality would be advantageous.

In order to simplify the solution space, as well as create more comprehensible results, an averaging technique was implemented to reduce the number of parameters required to describe the temporal flow rate or temperature differential. The high temporal resolution input data was downsampled and averaged in the process. Figure 5.15 is a plot demonstrating the temporal averaging technique. The data plotted in black are the flow rate fluctuations actually occurring in a cooling pond for the duration of a winter and is the 2,600-element time series referred to previously. The red line represents the average flow rate, calculated every 144 simulation hours. The averaging process reduces what was possibly a 2,600-dimensional space to a 20-dimensional space. With this shift, the swarm of particles that would be created would be defined by only 20 parameters, versus 2,600.
The time interval for averaging is determined by the characteristics of the problem set. A balance must be met between the number of parameters to optimize versus an adequate windowing to accurately represent the fluctuating flow rate. An increase in the number of windows used increases the dimensionality of the solution space to be searched. If too few windows are chosen, the averaged flow rate will not accurately reflect the actual conditions of the cooling pond and will yield poor simulation results. The balance was reached using user experience and intuition.

Once a window size is determined, the average flow rate is calculated over the amount of time contained within the average window and then assigned to each temporal point contained within that window. Each flow rate value, repeated for the entirety of a step (or window) in the plot, becomes a parameter for a given particle. For this particular example, the total amount of simulation time, divided into 144 hour increments, yields 20 different averaging windows (or steps). A particle for this application would be defined by 20 parameters, each of whose value would represent the flow rate at the corresponding
windowed time. Each iteration of a particle would have different values assigned to each of the 20 averaging windows derived from the success of the particle in its previous history and the swarm’s overall history. A similar method can be applied to the temperature differential across a cooling pond. For the test cases, the meteorological and temperature differentials were held constant at their observed values.

5.4.3 Convergence condition

To determine the validity of an ALGE simulation, the ice coverage estimations generated by ALGE were compared to ground and aerial-based image observations of ice coverage. The exclusion of the temperature comparisons from the swarm processes is supported in Sections 5.2.8 and 5.3. Originally, a swarm was considered to have converged if all of the personal best solutions achieved by each particle produced a metric value within $\pm 1 \cdot 10^{-4}$ of the best globally achieved RMS value. However, upon inspection of the progress of each swarm, it was decided that this original goal, considering the problem set being attempted, was a far too aggressive convergence parameter. The constraints for convergence were relaxed to allow convergence to be declared once all the best particle solutions achieved metric evaluations within $\pm 1 \cdot 10^{-2}$ of the global best.

5.4.4 Implementation of PSO-ALGE on computing cluster

Because of the parallel nature of PSO, the workflow to perform the optimization process was implemented on a high performance computing cluster. Process drivers were constructed to initiate ALGE runs as swarms of particles (See Appendix Q.2). For example, a swarm of 16 particles in this application would take the form of 16 separate ALGE runs, each initialized on a different processing node of the computing cluster. Each of the 16 simulations would be initialized with input parameters that were randomly selected based on the bounding parameters of the solution space. Each of the 16 simulations would complete and then be evaluated using the functional metric. The simulation which produces the closest match to the observed data, as determined by the metric, is designated the global best solution in the whole swarm. For the first generation, every solution represents its own personal best solution. Based on the positions in the solution space of the globally best achieved solution and the swarm’s personal best solutions, the velocity vectors are calculated using Equation 3.1. The velocity vector is added to each
solution’s current position in the solution space to create the next generation of input parameters.

Figure 5.16: Graphical representation of the PSO optimized ALGE algorithm as implemented on a computing cluster

5.5 Summary

This chapter walked the reader through the methodology applied to implement a PSO-driven ALGE optimization. The acquisition of all empirical data and the instruments used were detailed. In order to evaluate the success of a given ALGE simulation, a new approach to validation was developed to account for the ice formation now possible in the cold climate environment. Using the developed metric as the functional evaluation, ALGE was implemented using a PSO architecture on a high performance computing cluster. A custom set of analysis tools were created to process the results and interface with the PSO algorithm. The next chapter will outline the test cases created to pilot this methodology by applying it to real and synthetic data sets.
Chapter 6

Results

This section presents the results from the PSO optimization of ALGE. The optimization was tested for both the 2008-2009 and 2009-2010 winter data sets, as well as shorter simulation times to investigate the required validation interval and the swarm approach repeatability.

6.1 Initial optimization results

With each swarm there is a plethora of data produced that needs to be distilled to determine the success or failure of the simulations produced. An ideal success for a swarm would be for all solutions (or particles) to produce an average flow rate that matches the actual measured flow rate. A simulated flow rate that matches the observed average would produce simulated ice fractions that demonstrate high correlation with observed values.

Each of the swarms consisted of 16 different particles, or ALGE simulations, whose parameters were the windowed flow rate averages for the corresponding winter. A particle in a generation represents a set of flow rate parameters for a single ALGE simulation. Each particle had 20 parameters, each parameter is the flow rate for a segment of 144 simulation hours (approximately a week). For both winters, this definition resulted in a 20-dimensional solution space bound by an upper and lower limit of $25.0 \frac{m^3}{s}$ and $8.0 \frac{m^3}{s}$, respectively. These bounds were determined based on user interaction with the data set. In future applications these values would be characteristic of the problem set being solved. Each swarm was initialized to allow for up to 250 generations to be processed.

The ice fraction data that was used for the simulations of both the 2008-2009 and
2009-2010 winters is shown in Table 6.1. Duplicate days represent multiple observations recorded at different times for the given date.

<table>
<thead>
<tr>
<th>Date</th>
<th>Ice Fraction</th>
<th>Date</th>
<th>Ice Fraction</th>
<th>Date</th>
<th>Ice Fraction</th>
</tr>
</thead>
<tbody>
<tr>
<td>12/18/08</td>
<td>0.72</td>
<td>12/18/09</td>
<td>0.33</td>
<td>02/07/10</td>
<td>0.76</td>
</tr>
<tr>
<td>01/15/09</td>
<td>0.90</td>
<td>12/21/09</td>
<td>0.32</td>
<td>02/08/10</td>
<td>0.77</td>
</tr>
<tr>
<td>01/26/09</td>
<td>0.75</td>
<td>12/24/09</td>
<td>0.44</td>
<td>02/11/10</td>
<td>0.65</td>
</tr>
<tr>
<td>02/02/09</td>
<td>0.52</td>
<td>12/30/09</td>
<td>0.34</td>
<td>02/11/10</td>
<td>0.66</td>
</tr>
<tr>
<td>02/03/09</td>
<td>0.69</td>
<td>01/07/10</td>
<td>0.94</td>
<td>02/11/10</td>
<td>0.66</td>
</tr>
<tr>
<td>02/04/09</td>
<td>0.59</td>
<td>01/10/10</td>
<td>0.94</td>
<td>02/12/10</td>
<td>0.63</td>
</tr>
<tr>
<td>02/05/09</td>
<td>0.75</td>
<td>01/12/10</td>
<td>0.87</td>
<td>02/15/10</td>
<td>0.65</td>
</tr>
<tr>
<td>02/06/09</td>
<td>0.67</td>
<td>01/15/10</td>
<td>0.74</td>
<td>02/18/10</td>
<td>0.54</td>
</tr>
<tr>
<td>02/09/09</td>
<td>0.44</td>
<td>01/18/10</td>
<td>0.41</td>
<td>02/23/10</td>
<td>0.46</td>
</tr>
<tr>
<td>02/10/09</td>
<td>0.44</td>
<td>01/21/10</td>
<td>0.59</td>
<td>02/28/10</td>
<td>0.22</td>
</tr>
<tr>
<td>02/16/09</td>
<td>0.00</td>
<td>01/23/10</td>
<td>0.39</td>
<td>03/04/10</td>
<td>0.17</td>
</tr>
<tr>
<td>02/24/09</td>
<td>0.91</td>
<td>01/27/10</td>
<td>0.32</td>
<td>03/04/10</td>
<td>0.17</td>
</tr>
<tr>
<td>02/25/09</td>
<td>0.71</td>
<td>01/30/10</td>
<td>0.42</td>
<td>03/04/10</td>
<td>0.12</td>
</tr>
<tr>
<td>03/04/09</td>
<td>0.56</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>03/13/09</td>
<td>0.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 6.1: Observed ice fraction values for both the 2009-2009 and 2009-2010 winters

Using the ice metric evaluation, the 2008-2009 swarm converged after 18 generations and ran a total of 288 ALGE simulations. While actual individual computational time varied depending on the computing resource acquired for a given simulation, the average ALGE simulation required 7 hours to complete. To generate the results, approximately 2,016 hours of cumulative computational time were required, or 84 days. However, due to the parallel nature of PSO, 16 simulations were running simultaneously at any given time. The actual processing time was 126 hours, or 5.25 days. Analogously, the 2009-2010 swarm converged after 47 generations and ran a total of 752 ALGE simulations. These simulations represent 5,264 hours of cumulative computational time, or 219 days. Under the parallel processing paradigm, the actual processing time was approximately 329 hours, or 13 days.
6.1. Initial Optimization Results

6.1.1 Winter 2008-2009 Results

Shown in Table 6.2 are the simulated ice fraction results from the best achieving particle during its initial and final generation for the 2008-2009 swarm. These ice fractions are compared to the observed values and evaluated using a standard RMS metric at each generation. The last row of the table shows the initial and final RMS calculation for this particular particle in these generations. The metric values behaved as expected. The swarm was designed to converge on a solution producing the lowest metric value out of the entire population. However, the improvement in correlation between the initial optimization-driven ice fractions and the final values was small.

<table>
<thead>
<tr>
<th>Date</th>
<th>Observed</th>
<th>Initial</th>
<th>Final</th>
</tr>
</thead>
<tbody>
<tr>
<td>12/18/08</td>
<td>0.72</td>
<td>0.84</td>
<td>0.75</td>
</tr>
<tr>
<td>01/15/09</td>
<td>0.90</td>
<td>0.90</td>
<td>0.96</td>
</tr>
<tr>
<td>01/26/09</td>
<td>0.75</td>
<td>0.91</td>
<td>0.88</td>
</tr>
<tr>
<td>02/02/09</td>
<td>0.52</td>
<td>0.69</td>
<td>0.62</td>
</tr>
<tr>
<td>02/03/09</td>
<td>0.69</td>
<td>0.73</td>
<td>0.67</td>
</tr>
<tr>
<td>02/04/09</td>
<td>0.59</td>
<td>0.79</td>
<td>0.70</td>
</tr>
<tr>
<td>02/05/09</td>
<td>0.75</td>
<td>0.88</td>
<td>0.83</td>
</tr>
<tr>
<td>02/06/09</td>
<td>0.67</td>
<td>0.83</td>
<td>0.78</td>
</tr>
<tr>
<td>02/09/09</td>
<td>0.44</td>
<td>0.52</td>
<td>0.53</td>
</tr>
<tr>
<td>02/10/09</td>
<td>0.44</td>
<td>0.50</td>
<td>0.45</td>
</tr>
<tr>
<td>02/16/09</td>
<td>0.00</td>
<td>0.12</td>
<td>0.22</td>
</tr>
<tr>
<td>02/24/09</td>
<td>0.91</td>
<td>0.95</td>
<td>0.98</td>
</tr>
<tr>
<td>02/25/09</td>
<td>0.71</td>
<td>0.66</td>
<td>0.84</td>
</tr>
<tr>
<td>03/04/09</td>
<td>0.56</td>
<td>0.66</td>
<td>0.67</td>
</tr>
<tr>
<td>03/13/09</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>

Table 6.2: Simulated ice fractions from the best achieving particle’s initial and final generations for the 2008-2009 winter.

The reason for the small gain made by the optimization is evident when one examines where the swarm began and where it finished. The linear correlation between the simulated ice fractions and observed values, shown in Figure 6.1, were calculated for both the first
and final solutions. It is evident that the swarm did not converge on a significantly better solution because the initial solution was randomly initialized to a relatively good solution. The initial solution produced ice fractions that had high correlation with the observed values. The swarm did not have a lot of error to reduce in the solution. Additionally, despite having a very low RMS value for both the initial and final states of the simulation, the resulting “optimized” flow rate was a very poor candidate solution.

![Initial simulation ice fractions](image1.png) ![Final simulation ice fractions](image2.png)

Figure 6.1: Correlation between the observed and simulated ice fractions for the best particle in the first and final generations of the 2008-2009 winter simulations.

An ideal solution will produce an average flow rate that mirrors the average flow rate observed. Figure 6.2 shows two plots illustrating the average flow rate used to produce the first and final simulations for the best particle. As indicated by these plots, there is little, to no, similarity between the true averaged flow rate and the swarm produced flow rates, despite the low RMS values when comparing the simulated and observed ice fractions.
6.1. Initial optimization results

(a) Initial simulation flow rate  (b) Final simulation flow rate

Figure 6.2: Comparison of the averaged flow rate used to produce both the initial and final simulations for the swarm’s best solution. The true average, derived from known plant parameters, is shown in blue while the swarm results are shown in red. An ideal solution would produce an averaged flow rate matching the true average.

6.1.2 Winter 2009-2010 Results

Shown in Table 6.3 are the simulated ice fraction results from the best achieving particle during its initial and final generation for the 2009-2010 swarm. These ice fractions are compared to the observed values and evaluated using a standard RMS metric at each generation. The last row of the table shows the initial and final RMS calculation for this particular particle in these generations. Once again, only a small gain was made by the optimization. From the linear correlation between the simulated ice fractions and observed values, shown in Figure 6.3, it is evident that the swarm did not converge on a significantly better solution. The optimized solution did not offer much improvement over the initial, randomized solution. Again, an ideal solution will produce an average flow rate that mirrors the average flow rate observed. Below, in Figure 6.4, are two plots showing the flow rate used to produce the first and final simulations for the best particle.
As indicated by these plots, there is little, to no, similarity between the true averaged flow rate and the swarm produced flow rates.

<table>
<thead>
<tr>
<th>Date</th>
<th>Observed</th>
<th>Initial</th>
<th>Final</th>
<th>Date</th>
<th>Observed</th>
<th>Initial</th>
<th>Final</th>
</tr>
</thead>
<tbody>
<tr>
<td>12/18/09</td>
<td>0.33</td>
<td>0.53</td>
<td>0.47</td>
<td>02/07/10</td>
<td>0.76</td>
<td>0.89</td>
<td>0.79</td>
</tr>
<tr>
<td>12/21/09</td>
<td>0.32</td>
<td>0.36</td>
<td>0.36</td>
<td>02/08/10</td>
<td>0.77</td>
<td>0.96</td>
<td>0.87</td>
</tr>
<tr>
<td>12/24/09</td>
<td>0.44</td>
<td>0.30</td>
<td>0.34</td>
<td>02/11/10</td>
<td>0.65</td>
<td>0.83</td>
<td>0.61</td>
</tr>
<tr>
<td>12/30/09</td>
<td>0.34</td>
<td>0.94</td>
<td>0.65</td>
<td>02/11/10</td>
<td>0.66</td>
<td>0.86</td>
<td>0.56</td>
</tr>
<tr>
<td>01/07/10</td>
<td>0.94</td>
<td>0.79</td>
<td>0.71</td>
<td>02/11/10</td>
<td>0.66</td>
<td>0.78</td>
<td>0.51</td>
</tr>
<tr>
<td>01/10/10</td>
<td>0.94</td>
<td>0.91</td>
<td>0.96</td>
<td>02/12/10</td>
<td>0.63</td>
<td>0.79</td>
<td>0.48</td>
</tr>
<tr>
<td>01/12/10</td>
<td>0.87</td>
<td>0.80</td>
<td>0.90</td>
<td>02/15/10</td>
<td>0.60</td>
<td>0.69</td>
<td>0.37</td>
</tr>
<tr>
<td>01/15/10</td>
<td>0.74</td>
<td>0.61</td>
<td>0.67</td>
<td>02/18/10</td>
<td>0.54</td>
<td>0.63</td>
<td>0.26</td>
</tr>
<tr>
<td>01/18/10</td>
<td>0.41</td>
<td>0.57</td>
<td>0.56</td>
<td>02/23/10</td>
<td>0.46</td>
<td>0.47</td>
<td>0.11</td>
</tr>
<tr>
<td>01/21/10</td>
<td>0.59</td>
<td>0.59</td>
<td>0.54</td>
<td>02/28/10</td>
<td>0.22</td>
<td>0.41</td>
<td>0.11</td>
</tr>
<tr>
<td>01/23/10</td>
<td>0.39</td>
<td>0.78</td>
<td>0.68</td>
<td>03/04/10</td>
<td>0.17</td>
<td>0.35</td>
<td>0.09</td>
</tr>
<tr>
<td>01/27/10</td>
<td>0.32</td>
<td>0.61</td>
<td>0.36</td>
<td>03/04/10</td>
<td>0.17</td>
<td>0.34</td>
<td>0.63</td>
</tr>
<tr>
<td>01/30/10</td>
<td>0.42</td>
<td>0.96</td>
<td>0.82</td>
<td>03/04/10</td>
<td>0.12</td>
<td>0.32</td>
<td>0.58</td>
</tr>
</tbody>
</table>

Table 6.3: Simulated ice fractions from the best achieving particle’s initial and final generations for the 2009-2010 winter.

\[
\text{RMS Metric Value: INITIAL: 0.276 \quad \text{FINAL: 0.172}}
\]

Figure 6.3: Correlation between the observed and simulated ice fractions for the best particle in the first and final generations of the 2009-2010 winter simulations.
6.1. Initial optimization results

(a) Initial simulation flow rate

(b) Final simulation flow rate

Figure 6.4: Comparison of the averaged flow rate used to produce both the initial and final simulations for the swarm’s best solution during the 2009-2010 winter. The true average, derived from known plant parameters, is shown in blue while the swarm results are shown in red. An ideal solution would produce an averaged flow rate matching the true average.

6.1.3 Overall conclusions from initial simulations

When one examines how all the particles in the 2008-2009 and 2009-2010 swarms behaved, all the particles did converge to similar solutions while reducing the convergence parameter. Shown in Figures 6.5 and 6.6 are comparison plots illustrating all the particle flow rates for each swarm from both the beginning and end of the swarm’s life for the respective winters. Each individual red line is a single particle’s flow rate. The entire swarm of particle-specific flow rates are overlaid on top of one another other to demonstrate the initial spread in the data and the convergence to approximately the same flow rate. The true average flow rate is shown in blue. The initial generation of flow rates for both winters’ swarms spanned the entire range of possible flow rates and were random. As the swarms progressed, the solutions in each swarm converged to similar averaged flow rates, but not the true averages. If these swarms were to have performed perfectly, all of the
plotted red lines would have laid nearly, if not directly, on top of the blue average lines. The demonstrated behavior indicates that the swarms are performing as expected, but the model results being produced were not adequate.

Figure 6.5: Comparison of all the flow rates used to produce both the initial and final simulations for the entire swarm during the 2008-2009 winter. The true average, derived from known plant parameters, is shown in blue while the swarm results are shown in red. And ideal solution would have all the swarm produced flow rates converging to match the true average flow rate.

A theory was formed that the inability of the PSO-ALGE system to converge on reasonable solutions using these swarms was tied to the initialization of the swarm and the downstream impacts on the thermodynamic conditions being modeled. The initialization states potentially hindered the swarm’s ability to recover to a reasonable flow rate solution through the evolution of the population because the starting flow rates were so erratic. The radical change from one temporal averaging window to the next in the initial flows imply that the flow rate of heated water effluent from the power plant changed dramati-
6.1. Initial optimization results

(a) Initial simulation flow rate  
(b) Final simulation flow rate

Figure 6.6: Comparison of all the flow rates used to produce both the initial and final simulations for the entire swarm during the 2009-2010 winter. The true average, derived from known plant parameters, is shown in blue while the swarm results are shown in red. An ideal solution would have all the swarm produced flow rates converging to match the true average flow rate.

cally after being held constant for a considerable amount of time. Not only was the change dramatic, but it spanned the entire range of possible flow rates. While in reality the power plant being modeled did have rapid fluctuations in flow rate, the total range of flow rates was much smaller and the fluctuations were on an hourly scale. The true average rate did not fluctuate wildly or widely. The swarms were initialized in states that did not mimic true ground conditions and created difficult and potentially unstable thermodynamic conditions. Additionally, the possible range of flow rates averaged to approximately the true average flow rate in the cooling pond. Because of the work completed by Garrett et al [12] and described in Section 5.2.8 that demonstrated the strong correlation between simulated ice fraction and average seasonal heat load, the average answer from these erratic
flow rates would still generate an ice correlation with a relatively low RMS error.

### 6.2 Swarm initialization impact

In order to investigate the theory that the swarm initialization conditions had a significant impact on the overall outcome, a 2009-2010 swarm was re-run under different start-up conditions. The initial flow rates were forced to conform to a distribution of lower mean and markedly smaller variance. The hypothesis supporting this decision was that a more controlled initialization state would allow each of the particle’s candidate ALGE solutions to spin up to a reasonable thermodynamic state without introducing a memory issue into the ice formation that would ultimately be hard to recover from. Shown in Figure 6.7 are the initialization states for each particle in the re-run swarm.

![Initial simulation flow rates](image1)

![Final simulation flow rates](image2)

(a) Initial simulation flow rates  
(b) Final simulation flow rates

Figure 6.7: Initial and final flow rate distribution for the trial swarms for the 2009-2010 data set. This swarm was initialized at a lower mean flow rate and a smaller variance to help mitigate the initialization failures of the previous full winter simulations.

Both of these swarms converged to successful solutions and performed noticeably better than initial simulations. Shown in Figure 6.8 are the differences in linear correlation between the simulated and observed ice fractions for the initial and final swarms under
6.3 Sparse validation data theory

As seen in these results, the initialization states were significantly different than the original swarm attempts. All initial flow rates were forced to a mean of approximately 8 \( \frac{m^3}{s} \) with a variance of approximately 1 \( \frac{m^3}{s} \). This average flow rate was significantly lower than the true known average, denoted by the dotted blue line in Figure 6.7. The final converged solution, while not identical to the known flow rate, produced an average flow rate within approximately 12%. An adequate solution is required to be within 10% of the true value [10].

6.3 Sparse validation data theory

A second trial was run to investigate the effect of sparser validation data on the overall performance of the swarm and the quality of the simulations produced. For this application, the time interval between validation data points is analogous to the time difference between image acquisitions. Using the same setup conditions implemented for the pre-
vious re-run 2009-2010 data set, a second re-run trial swarm was created using a subset of the validation data. The use of the subassembled validation data set created a condition where there were extended periods of time between collection intervals. While the time between collection points for the original dataset varied from 1 hour to a few days, this subassembled data set was forced to have a minimum of 12 days in between validations points. The resulting flow rates and ice fraction correlations are shown in Figures 6.9 and 6.10, respectively.

Figure 6.9: Initial and final flow rate distribution for the trial swarms for the 2009-2010 data set with a sparser validation data set.
As seen in Figure 6.10, the swarm was able to converge onto a solution that produced good ice fraction correlations. However, the average flow rates (shown in Figure 6.9) that generated those good correlations deviated from the true average flow rate by approximately 29%. The deviation from the implied performance from the metric and the actual performance indicated by the average flow rate is potential evidence of a “breaking point” in the metric. The ramifications of a “breaking point” translate to an insensitivity in the metric to detect a convergence failure at a given validation interval. The effect of this observation was the creation of a sparse validation data theory to describe the potential limitation of the approach as a practical application. This finding was not unexpected.

Insight into the existence of a metric weakness would be valuable due to the likelihood of intermittent data collection in real world operations. In the full winter swarm implementations, each windowed average flow rate is allowed to vary throughout the duration of a simulation, irregardless of the presence of observation values. An assumption is made that the flow rate at a point in time prior to and after the validation point is bound by the governing thermodynamic equations and laws inherent to the ALGE model. While this assumption most likely holds true for a system with very little or gradual variation, it may not remain true for a dynamic system that can change rapidly on an hourly time scale with sparse validation data sets. The full winter ALGE simulations were composed of approximately 2,600 hours worth of simulation time. While environmental and thermodynamic
properties changed at the hourly rates, each winter simulation was supplied 20-30 points in time, collected at inconsistent intervals, to validate the model state. Given the high variability in the plant operating parameters, there was a desire to analyze the conditions where the temporal interval between validation data points was sparse and whether it would allow for too much freedom in parameter optimization of a highly dynamic system.

6.3.1 Sparse validation data experiment

An experimental approach was designed and executed to determine the effect of sparse image validation data. To reduce as much uncertainty as possible, shorter simulations were created using both collected and manipulated plant and meteorological conditions. The sparse validation theory was tested by measuring a swarm’s ability to converge to a solution at varying validation intervals under constant flow rate conditions. The validation interval refers to the simulation time allowed to lapse between two points in the simulation period where the simulated ice fraction is compared to the image-derived, observed ice fraction. Again, in reality this validation data is derived from acquired imagery and the validation interval refers to the time lapse between collection. For these experiments, the image-derived observations were replaced with simulated ice fractions. Constant flow rate conditions imply that the flow rate of the simulated facility was held constant for the duration of the simulation. The meteorological conditions act as the only influencing factor on ice formation.

Nine swarms were started using the same initial conditions, however, the validation intervals were varied from every 1 hour to every 576 hours (24 days). The original experimental design did not anticipate the requirement to explore validation intervals beyond 72-hours and resulted in a limiting simulation duration of two weeks. Preliminary results indicated a need to investigate longer validation intervals and as a result an additional constant flow rate condition simulation was created. For the first simulation, the heat effluent flow rate was held at a constant $20.7 \frac{m^3}{s}$ and the air temperature was manipulated to induce two “melt and freeze” periods over a two week timeframe in December of 2008. The second constant flow rate condition simulation was constructed, using the same constant $20.7 \frac{m^3}{s}$ flow rate, however it spanned four weeks of simulation time during February 2009. Additionally, the air temperature was not manipulated to force “melt and freeze” periods during longer the simulation. The hourly ice fractions resulting from the simulations were treated as the observed ice fractions, or truth, and replaced the actual
acquired image-derived ice fractions for the PSO attempts. It was assumed that at more condensed validation intervals, the PSO approach would perform better. Similar to the full winter simulations, the temporal averaging technique was employed to limit the dimension of the solution space for each swarm. To avoid the random initialization of the first swarm solutions to good solutions, starting flow rates for all instances were forced to have a significantly lower mean rates.

### 6.3.2 Sparse validation data results

Shown in Figures 6.11-6.19 are the differences in the linear correlations of observed and simulated ice fractions for both the initial and final solutions generated by the PSO swarms for each of the different validation intervals. The red plotted line and the associated correlation data overlaid in the plot are derived from the correlation between the simulated and observed data. The green dotted line is a perfect one-to-one line with a zero intercept. These two lines are displayed to illustrate that while the correlation can be high for a data set, it does not necessarily mean a one-to-one correspondence. The only variation is the validation interval used in the convergence metric, which would represent the collection interval between image observations.

![Graphs showing initial and final ice fraction correlations](image)

(a) Initial ice fraction correlation  
(b) Final ice coverage correlation

Figure 6.11: Comparison of initial and final ice fraction correlations for the 1-hour validation interval.
CHAPTER 6. Results

Figure 6.12: Comparison of initial and final ice fraction correlations for the 12-hour validation interval.

(a) Initial ice fraction correlation  
(b) Final ice coverage correlation

Figure 6.13: Comparison of initial and final ice fraction correlations for the 24-hour (1 day) validation interval.

(a) Initial ice fraction correlation  
(b) Final ice coverage correlation
6.3. Sparse validation data theory

Figure 6.14: Comparison of initial and final ice fraction correlations for the 48-hour (2 day) validation interval.

Figure 6.15: Comparison of initial and final ice fraction correlations for the 72-hour (3 day) validation interval.
(a) Initial ice fraction correlation  
(b) Final ice coverage correlation

Figure 6.16: Comparison of initial and final ice fraction correlations for the 120-hour (5 day) validation interval.

(a) Initial ice fraction correlation  
(b) Final ice coverage correlation

Figure 6.17: Comparison of initial and final ice fraction correlations for the 168-hour (7 day) validation interval.
6.3. Sparse validation data theory

Figure 6.18: Comparison of initial and final ice fraction correlations for the 288-hour (12 day) validation interval.

Figure 6.19: Comparison of initial and final ice fraction correlations for the 576-hour (24 day) validation interval.

Each of the validation intervals perform satisfactory under the constant flow rate conditions, however, the convergence metric alone does not indicate how successfully a given swarm performed. Shown in Figures 6.20-6.28 are comparisons between the initial and
final swarm flow rates produced for each validation interval.

Figure 6.20: Swarm flow rates for constant flow rate conditions and 1-hour validation interval.

Figure 6.21: Swarm flow rates for constant flow rate conditions and 12-hour validation interval.
6.3. Sparse validation data theory

Figure 6.22: Swarm flow rates for constant flow rate conditions and 24-hour (1 day) validation interval.

Figure 6.23: Swarm flow rates for constant flow rate conditions and 48-hour (2 day) validation interval.
Figure 6.24: Swarm flow rates for constant flow rate conditions and 72-hour (3 day) validation interval.

Figure 6.25: Swarm flow rates for constant flow rate conditions and 120-hour (5 day) validation interval.
6.3. Sparse validation data theory

Figure 6.26: Swarm flow rates for constant flow rate conditions and 168-hour (7 day) validation interval.

Figure 6.27: Swarm flow rates for constant flow rate conditions and 288-hour (12 day) validation interval.
As the validation interval grows, the average converged swarm flow rates drifts from the true average flow rate, beginning at the 288-hour (12 day) interval. The dispersion pattern in the data, away from the true, desired average, is directly related to the freedom allowed to the flow rate as the validation interval became larger. Segments of the flow rate that remain unchecked for longer periods of simulation time are susceptible to larger variance due to the randomness inherent in the PSO process. The natural randomness of PSO allows a swarm to adequately search a large solution space. However, if a parameter has no data for comparison and is only weakly related or responsive to the behavior of neighboring parameters, then it will merely behave randomly. Because parameters in this application are actual flow rates, this randomness, and its associated increase with lack of validation, directly affect the ice formation and ultimately the outcome of the candidate solution.

It is reasonable to assume that there are bounds within which the validation interval can shift and the resulting flow rates will remain restricted by the thermodynamic and fluid dynamic rules imposed by the ALGE model. Shown in the data presented above in Figures 6.11-6.28 and Table 6.4, that this boundary occurs between the 168- and 288-hour (7-12 days) intervals. At the 1-hour through the 168-hour (7 day) intervals, the convergence metric and the resulting flow rates all behaved as expected and produced satisfactory ice coverage fractions and simulation flow rates. These simulations were considered to have
6.4 Swarm Repeatability

The randomness inherent to the PSO process generated a question as to the repeatability of a swarm. In order to determine if the initialization conditions of a given swarm could be relied upon to create the same convergence parameters repeatedly, the 12-hour interval swarm, under constant flow rate conditions, was repeated 12 times. The initial and final

<table>
<thead>
<tr>
<th>Validation Interval [Hours]</th>
<th>Initial Corr.</th>
<th>Final Corr.</th>
<th>Initial RMS</th>
<th>Final RMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.623</td>
<td>1.000</td>
<td>0.760</td>
<td>0.005</td>
</tr>
<tr>
<td>12</td>
<td>0.715</td>
<td>0.988</td>
<td>0.665</td>
<td>0.006</td>
</tr>
<tr>
<td>24 (1 day)</td>
<td>0.689</td>
<td>0.999</td>
<td>0.673</td>
<td>0.017</td>
</tr>
<tr>
<td>48 (2 days)</td>
<td>0.704</td>
<td>0.997</td>
<td>0.660</td>
<td>0.049</td>
</tr>
<tr>
<td>72 (3 days)</td>
<td>0.717</td>
<td>1.000</td>
<td>0.847</td>
<td>0.020</td>
</tr>
<tr>
<td>120 (5 days)</td>
<td>0.790</td>
<td>1.000</td>
<td>0.631</td>
<td>0.022</td>
</tr>
<tr>
<td>168 (7 days)</td>
<td>1.000</td>
<td>1.000</td>
<td>0.558</td>
<td>0.034</td>
</tr>
<tr>
<td>288 (12 days)</td>
<td>0.681</td>
<td>0.926</td>
<td>0.626</td>
<td>0.306</td>
</tr>
<tr>
<td>576 (24 days)</td>
<td>1.000</td>
<td>1.000</td>
<td>0.904</td>
<td>0.418</td>
</tr>
</tbody>
</table>

Table 6.4: Ice coverage correlation values and RMS error metrics between the simulated and observed ice fractions. These values are representative of the initial and final states of the converged solutions.

behaved as expected because the resulting simulated average flow rates were within an acceptable amount of error from the true average. At the 288-hour (12 day) and 576-hour (24 day) intervals, the solutions converged onto average flow rates that are significantly displaced from the true average flow rates. While the RMS metric values for these last two intervals continue to indicate that the swarms had produced adequate simulation parameters, the yielded flow rates had larger variance and no longer recreated the true average flow rates within an acceptable margin of error. An argument can be made that the “breaking point” of this approach actually is dependent on the acceptable error in the flow rate estimation. However, the observed variation in flow rate at this boundary point, coupled with the fact that the metric was insensitive to the poorer results, further illustrates that the “breaking point” occurs between the aforementioned hourly intervals. Additionally, this corroborates the observed results from the subsetted full winter run discussed in Section 6.2 and shown in Figures 6.9 and 6.10. The simulated data used to generate those results had a validation set with an interval minimum of 12 days between data points and resulted in less than adequate flow rate estimates.

6.4 Swarm Repeatability

The randomness inherent to the PSO process generated a question as to the repeatability of a swarm. In order to determine if the initialization conditions of a given swarm could be relied upon to create the same convergence parameters repeatedly, the 12-hour interval swarm, under constant flow rate conditions, was repeated 12 times. The initial and final
RMS error values between simulated and observed ice fractions for each swarm repetition is shown below in Table 6.5.

<table>
<thead>
<tr>
<th>Swarm Num.</th>
<th>Initial</th>
<th>Final</th>
<th>Num. of Gens.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.706</td>
<td>1.000</td>
<td>98</td>
</tr>
<tr>
<td>2</td>
<td>0.715</td>
<td>1.000</td>
<td>98</td>
</tr>
<tr>
<td>3</td>
<td>0.711</td>
<td>1.000</td>
<td>98</td>
</tr>
<tr>
<td>4</td>
<td>0.710</td>
<td>1.000</td>
<td>98</td>
</tr>
<tr>
<td>5</td>
<td>0.691</td>
<td>1.000</td>
<td>98</td>
</tr>
<tr>
<td>6</td>
<td>0.715</td>
<td>0.988</td>
<td>31</td>
</tr>
<tr>
<td>7</td>
<td>0.698</td>
<td>0.998</td>
<td>26</td>
</tr>
<tr>
<td>8</td>
<td>0.705</td>
<td>1.000</td>
<td>22</td>
</tr>
<tr>
<td>9</td>
<td>0.698</td>
<td>0.997</td>
<td>19</td>
</tr>
<tr>
<td>10</td>
<td>0.704</td>
<td>0.999</td>
<td>29</td>
</tr>
<tr>
<td>11</td>
<td>0.726</td>
<td>1.000</td>
<td>20</td>
</tr>
<tr>
<td>12</td>
<td>0.711</td>
<td>0.998</td>
<td>16</td>
</tr>
</tbody>
</table>

Table 6.5: Initial and final RMS error between simulated and observed ice fractions for the repeatability swarms. The number of generations indicates the number of swarm generations required to achieve convergence.

Each swarm instance was able to successfully converge onto accurate ice coverage estimates. It can be seen that the first group of swarms took considerably longer to converge (98 generations versus 20-30). The discrepancy between these convergence times was the result of an aggressive convergence condition for the first few swarms. Initially the swarms 1-5 were run and only allowed to converge if all of the personal best solutions achieved by each particle produced a metric value within $\pm 1 \cdot 10^{-4}$ of the best globally achieved RMS value. This was a mistake during the setup of the swarms for reasons addressed in Section 5.4.3. The constraints for convergence were relaxed to allow convergence to be declared once all of the best particle solutions achieved metric evaluations within $\pm 1 \cdot 10^{-2}$ of the global best for swarms 6-12. With this difference taken into account, these results indicate that the swarm results are reliable and repeatable. For completeness, the accompanying flow rate and ice coverage correlation plots are included in Appendix O.

6.5 Summary

This chapter presented results from several different experiments aimed at exploring the effectiveness and limitations of the PSO-ALGE architecture. Original results indicated that satisfactory results had a dependency on the initialization parameters the swarm
conditions. Subsequent swarm runs led to an investigation into the impact varying validation intervals had on simulation performance and accuracy. These investigations yielded the identification of a “breaking point” at which the data collection interval becomes too wide to expect adequate simulation results. Additionally, the repeatability of a swarm applied to ALGE was tested and determined to be reliable.
Chapter 7

Summary and Conclusions

The ultimate motivation for the work presented here was to improve the accuracy of the ALGE thermodynamic model when modeling cooling ponds in cold climates through the introduction of a novel parameter optimization technique, particle swarm optimization. The introduction of cold climate environmental possibilities (ice and snow) creates a very complex environment to both model and measure. The employed optimization technique used image-derived observables to drive the evolution of the candidate solutions to final solutions. The final workflow built to achieve this goal involved an implementation of a PSO-driven ALGE modeling environment running on a computing cluster.

In support of this effort, a two-year data campaign was planned, managed, and executed to collect a database of observations that would be useful for several facets of the effort. In order to accomplish the data campaigns several hardware systems were either improved or designed and built from scratch for deployment at the data validation site in Midland, MI. The WASP sensor was upgraded to include on-board blackbodies for calibration needs. A large effort was undertaken to create a calibration workflow for the WASP sensor’s MWIR and LWIR channels. The resulting tool set provides users with automated and headless command-line tools for producing radiance images for both channels, calibrated on a pixel-by-pixel basis. In-water buoys, a weather station, and a roof-mounted oblique imaging system were designed, deployed, and maintained at the collection site. A methodology was created for extracting ice conditions from pond imagery acquired by the oblique imaging system.

With the collection of all the various data sources, a PSO-ALGE architecture was created and stood up on RIT’s Research Computing Cluster to run ALGE using all the
collected data. The resulting system is a combination of IDL and bash scripts which execute a fully parallelized, file-based, PSO application for the ALGE model. In order to interface with the swarm of ALGE simulations both during execution and after convergence, another set of tools was created for monitoring swarm performance while actively running, as well as processing the final swarm results down to a series of plots, images, and text reports. These tools allow the user to not only execute an ALGE instance from the command line, but also to determine mid-swarm if the evolution of the potential solutions is reasonable and to analyze final results.

Prior to executing full ALGE swarms inside the cluster framework, an appropriate fitness metric had to be identified that would determine if a particular set of simulation parameters were performing better or worse than previous attempts. The two potential observables that could serve as validation data were the thermal distribution in the open water areas of the cooling pond or the amount of ice coverage present on the body of water. Initially, because of the complex thermodynamic conditions present in the environment, it was thought that thermal data would be a required observable. However, after experiments described in this work and supported by work by Garrett et al. [12], it was determined that ice coverage is an effective indicator of current plant operating conditions. This outcome by itself was a significant finding. The removal of the thermal data from collection requirements alleviates the need for a thermal imaging system to monitor a site of interest. As a result, a visible imaging system can be used and generate adequate validation data. However, this assumption requires either a priori knowledge of the temperature differential between the cooling pond intake and outflow points or the addition of this differential as an optimizing parameter.

Upon completion of all the processing tools, functional metrics, and data collection, several experiments were completed to determine the effectiveness and limitations of the PSO-ALGE approach. It was determined, that given a reasonable initialization state, PSO could adequately drive ALGE simulations to an appropriate flow rate solution. In order to marry the PSO approach to the ALGE model, a new method was developed for incorporating flow rate data as optimization inputs using temporally-based window averaging. After analyzing initial swarm results, it became of interest to determine if the interval between imagery collection times, or validation data intervals, would affect the sensitivity of the metric. It was found that this “breaking point” occurred at approximately 12 days. This interval helps define the concept of operations for a functional use of this
technique for a real world application. Additionally, the repeatability of these swarm results were investigated and determined to be excellent. A swarm was initialized 12 times with the same start up parameters. All 12 instances converged on the correct solution.

7.1 Recommendations for future work

The effort and results described in this work do not answer all the questions related to trying to effectively model a complex simulation such as the one outlined here. There are several areas that warrant recommendations for further work. These areas and their associated suggestions are listed below.

- **Particle Swarm Optimization:** Because of the nature of PSO and the lack of theoretical basis as to why it is able to converge to solutions in these complex solution spaces, it would be worthwhile to investigate further tuning of the PSO operational parameters. Values chosen for the parameters were taken from literature where they were documented as successful for other problem sets. It is possible these values could be tuned to produce faster convergence rates or better flow rate estimates.

- **Temporal Inputs for Optimization:** The temporal averaging technique created and implemented helped distill a large solution space into a manageable size, while reflecting the real world limitations of *a priori* information available to a user. It would have been much simpler to infer a baseline temporal flow rate and have a multiplicative factor be the optimized parameter. However, in real world applications, there would be no baseline flow rate estimate and its variations, more than likely, wouldn’t be multiplicative. It would be useful to investigate the development of a parameterize model for a facility flow rate that were driven by the actual function of the plant. For example, if a power plant’s power production rate was correlated to weather events (cold winters influence heating usage, hot summers influence amount of A/C use) then perhaps a model could be developed that incorporated these trends. If such a model was created, the variables controlling it’s evaluation would become the optimization parameters that define the PSO particle.

- **Optimization parameters:** The only variable investigated for this work was the flow rate. It would be worthwhile to apply the same approach to the meteorological
conditions, temperature differential on the ground, and the ice material and radiative properties.

- **Observable ice extent:** Only the amount of ice coverage was used as an input parameter to the process. It would be interesting to investigate whether the ALGE model was not only accurately modeling the amount of ice present on the pond, but where in the pond the ice was forming.
Bibliography


Appendix A

Radiation propagation in the thermal region

In order to remotely retrieve the temperature of a desired target, the radiant energy emitted from a target needs to be observed by a sensor that is sensitive in the thermal infrared region of the electromagnetic spectrum. Energy collected at the sensor focal plane is a combination of several sources, including the radiant energy originating from the target itself, the surrounding environment, and the atmosphere between the sensor and the ground plane. In order to extract a particular target’s absolute temperature, the observed signal needs to be deconstructed into its contributing parts and analyzed. The retrieval of a target’s absolute temperature is a difficult task due to the complex nature and interdependencies of the contributing elements in the total observed energy signal. The following section reviews the concepts behind the physical properties and observables in the thermal infrared region of the electro-magnetic spectrum [22].

A.1 Self-emittance

The amount of energy present within a material is described by a material’s temperature. Every material on earth sits at a temperature greater than absolute zero and will radiate and absorb energy within its environment in an attempt to reach thermodynamic equilibrium with its surroundings. The amount of energy radiated or absorbed by said material during interactions with its environment is a function of both the material’s temperature,
CHAPTER A. Radiation propagation in the thermal region

$T$, and emissivity, $\epsilon$.

A.2 Planck’s equation

An ideal material or surface is characterized as a blackbody when it absorbs and re-emits radiation at all wavelengths with perfect efficiency. Derived by Max Planck [21], the Planck blackbody radiation equation describes the spectral radiance, $L_{BB}(\lambda, T)$, emitted from a blackbody radiator into a solid angle about the blackbody’s surface. Planck’s equation is defined as

$$L_{BB}(\lambda, T) = \frac{2hc^2}{\lambda^5} \frac{1}{e^{\frac{hc}{\lambda kT}} - 1} \left[ \frac{W}{m^2sr\mu m} \right], \quad (A.1)$$

<table>
<thead>
<tr>
<th>h</th>
<th>Planck’s constant</th>
<th>$6.6256 \cdot 10^{-34} [J \cdot s]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>speed of light in vacuum</td>
<td>$2.9979 \cdot 10^8 [m/s]$</td>
</tr>
<tr>
<td>k</td>
<td>Boltzmann constant</td>
<td>$1.3807 \cdot 10^{-23} [J/K]$</td>
</tr>
<tr>
<td>$\lambda$</td>
<td>wavelength of emission</td>
<td>[$\mu m$]</td>
</tr>
<tr>
<td>T</td>
<td>absolute temperature</td>
<td>[K].</td>
</tr>
</tbody>
</table>

Table A.1: Equation variables and constants for Planck’s equation

Investigation of the equation terms shows that the radiance emitted by a blackbody is dependent on both the wavelength of emission as well as the target temperature. If the temperature of a blackbody was set to several fixed values, a family of blackbody curves would result. When plotted comparatively (Figure A.1), it is clear that as the temperature of a material increases, the radiance emitted by the material increases and the peak wavelength shifts towards shorter wavelengths.

Upon further analysis of a family of blackbody curves, it is demonstrated that a Planckian energy distribution is a well-behaved function and has only one maximum value. Solving for the zero-point of the first derivative of a distribution produces the wavelength at which the blackbody emits the maximum amount of radiant energy. Referred to as Wien’s displacement law, this particular solution describes the relationship between an object’s temperature and the wavelength at which the object exhibits maximum radiance. Mathematically speaking, as an object’s temperature increases, the overall shape of the
Figure A.1: Blackbody energy distributions for blackbodies sitting at 5800K, 3000K, and 300K. A solar spectra is compared to the energy distribution of a blackbody sitting at 5800K [22].

energy distribution remains the same, however, the peak wavelength for emittance is displaced on the plot towards shorter wavelengths. The mathematical expression for Wein’s displacement law is shown below in Equation A.2, where $A = 2897.768 [\mu m K]$,

$$\lambda_{peak} = \frac{A}{T} \ [\mu m].$$

Earth-based materials have an average temperature of 300 K. Through an application of Wein’s law, the resulting peak wavelength for radiant emission is in the longwave infrared region of the spectrum, approximately 10µm. This peak wavelength is advantageously located in the middle of an atmospheric transmission window (Figure A.2) and has dictated the development of thermal sensors using materials sensitive to radiant emission in the 8 – 14µm range.

### A.3 Emissivity

Materials exist in nature that behave similarly to blackbodies, but not identically. In order to describe how blackbody-like a material is, a ratio is calculated to compare the actual spectral emission from an object at a particular temperature to that of a perfect
blackbody’s spectral emission at the same temperature. This ratio is called the material’s emissivity, \( \epsilon(\lambda) \), and is shown in equation A.3.

\[
\epsilon(\lambda) = \frac{L(\lambda, T)}{L_{BB}(\lambda, T)} \quad (A.3)
\]

The resulting value is a unitless number with a value from 0 to 1. A perfect blackbody would have an emissivity of 1 and be a perfect emitter and absorber at a given temperature. Materials that exhibit emissivity values less than 1 and maintain the same value across all spectral regions are described as graybodies. Selective radiators have a wavelength-dependent emissivity [22].

### A.4 Conservation of energy and Kirchoff’s law

Emissivity is a material-dependent property. Similarly, absorbivity, transmittance, and reflectivity, are properties which describe how a given material interacts with energy in its environment. All these properties are intrinsic to the material. Incoming irradiance, \( E_i \), that falls incident on an object is transformed with respect to these three material properties. It is this alteration that distinguishes one material from another [22].

The absorbivity, \( \alpha(\lambda) \), of a material describes the material’s ability to turn incident energy into an alternative form of energy, \( E_a \) (i.e. thermal or kinetic energy). This
A.4. Conservation of energy and Kirchoff’s law

property is described using a ratio of incoming radiant energy to the produced alternative energy (Equation A.4).

\[ \alpha(\lambda) = \frac{E_a}{E_i} \quad (A.4) \]

A material’s reflectivity, \( r(\lambda) \), describes the ability of the material to change the direction of incident radiant energy and send it back into the above hemisphere. Reflectivity is described as the ratio of incoming irradiance to the irradiance leaving the material’s surface, \( E_r \) (Equation A.5).

\[ r(\lambda) = \frac{E_r}{E_i} \quad (A.5) \]

The transmittance, \( \tau(\lambda) \), of a material is defined as the capability of the material to allow energy to propagate through itself. Mathematically this property is represented by the ratio of incoming irradiance to the irradiance leaving the opposite side of the material, \( E_t \) (equation A.6).

\[ \tau(\lambda) = \frac{E_t}{E_i} \quad (A.6) \]

By the law of conservation of energy, when radiant energy comes in contact with an object, it is either absorbed, transmitted, or reflected. The unitless quantities of absorbtivity \( \alpha(\lambda) \), transmittance \( \tau(\lambda) \), and reflectivity \( r(\lambda) \), must sum to unity to satisfy this condition (Equation A.7).

\[ \alpha(\lambda) + r(\lambda) + \tau(\lambda) = 1 \quad (A.7) \]

Kirchoff’s Law states that, by definition, the absorbtivity of a material is equal to the emissivity when the material is at thermal equilibrium [24]. In practical terms this statement declares that good absorbers are also good emitters and the absorbtivity term in Equation A.7 can be substituted with emissivity as shown in Equation A.8.

\[ \epsilon(\lambda) + r(\lambda) + \tau(\lambda) = 1 \quad (A.8) \]

Furthermore, a blackbody would exhibit zero transmission and zero reflectivity of incident energy due to the defining characteristic that all energy absorbed is then re-emitted. Therefore, for a blackbody the transmittance is zero and Equation A.8 is reduced
to Equation A.9.

\[ \epsilon(\lambda) = 1 \]  
(A.9)

### A.5 Atmospheric and background effects

Similar to objects on the ground whose temperature is of interest, all matter between or near the object and sensor (i.e. the air column and background objects) will sit at an absolute temperature above zero and therefore will emit energy. As a result, the collected energy at the focal plane is a combination of several sources of energy taking different paths to enter the sensor. Figure A.3 illustrates all self-emissive sources and their related path to the sensor within a scene, including the target itself [22].

![Figure A.3: Thermal energy paths from potential sources of interest. Path A photons are photons emitted by the atmosphere and radiated directly into the sensor. Path B photons are emitted from some background object, fall onto the target, and are then reflected back towards the sensor. Path C photons are photons emitted by the target of interest. Path D photons are emitted by the atmosphere, towards the target, reflect of its surface, and are collected at the sensor.](image)

Radiance emitting from the target of interest, or energy traveling along path C, is the most important source of radiance. Photons traveling along path B are emitted from some background object, fall onto the target, and are then reflected back towards the sensor.
These photons are grouped into the background radiance component. The atmosphere also emits photons, some of which are radiated directly into the sensor (path A). This radiant component is referred to as the upwelling radiance. Photons traveling along path D, which are emitted from the atmosphere, towards the target, reflect off its surface, and are collected at the sensor. This radiant component is referred to as the downwelling radiance.

A.6 Atmospheric transmission

As radiant energy travels through the atmosphere from the target to the sensor, some energy from the self-emissive sources will be lost due to absorption and scattering. The total effect of these two phenomena is modeled by the total atmospheric transmission.

Component molecules of the atmosphere absorb a fraction of the radiant energy and translate it into another form of energy (i.e. thermal energy). The amount of energy a layer of atmosphere can absorb is dependent on the atmospheric constituents, their associated concentrations, temperature, and pressure and is calculated using Equation A.10 [15].

\[ \tau(\lambda) = e^{-C(\lambda)_{\alpha} m z} \] (A.10)

Equation A.10 represents the atmospheric transmission in terms of concentration-depth absorption. The absorption cross-section, \( C(\lambda)_{\alpha} \), is the effective size of a molecule to the photon flux at that wavelength. Additionally, \( m \) is the number density or number of molecules per unit volume and \( z \) is the path length.

Energy is scattered when it is forced out of the propagating beam of energy through interaction with molecules it encounters along its path of travel. Scattering is classified into three types: Rayleigh scattering, Mie scattering, and nonselective scattering. Rayleigh scattering describes an event where an energy wave interacts with spherical particles significantly smaller than the wavelength of the incident flux. Mie scattering occurs when the wavelength of the incident flux is the same order of magnitude in size of the particles. Nonselective scattering occurs when the wavelength of the incident flux is significantly smaller than the size of the particles.

All specified types of transmission loss sum together to produce a total value for transmission for a given atmosphere, shown in Equation A.11.
\[ \tau(\lambda) = e^{-(\beta(\lambda)_\alpha + \beta(\lambda)_r + \beta(\lambda)_m + \beta(\lambda)_{ns})z} = e^{-(\beta(\lambda)_{ext})z} \]  
\hspace{1cm} \text{(A.11)}

<table>
<thead>
<tr>
<th>\beta(\lambda)_\alpha</th>
<th>\text{fractional amount of energy lost due to absorption per unit length}</th>
</tr>
</thead>
<tbody>
<tr>
<td>\beta(\lambda)_r</td>
<td>\text{fractional amount of energy lost due to Rayleigh scattering}</td>
</tr>
<tr>
<td>\beta(\lambda)_m</td>
<td>\text{fractional amount of energy lost due to Mie scattering}</td>
</tr>
<tr>
<td>\beta(\lambda)_{ns}</td>
<td>\text{fractional amount of energy lost due to Nonselective scattering}</td>
</tr>
<tr>
<td>\beta(\lambda)_{ext}</td>
<td>\text{summation of all scattering and absorption coefficients}</td>
</tr>
<tr>
<td>z</td>
<td>\text{path length of propagating beam}</td>
</tr>
</tbody>
</table>

Table A.2: Equation variables for total transmission

In Equation A.11 the variables represent the following physical parameters: \( \beta(\lambda)_\alpha \) is the fractional amount of energy lost due to absorption per unit length, \( \beta(\lambda)_r \) is the fractional amount of energy lost due to Rayleigh scattering, \( \beta(\lambda)_m \) is the fractional amount of energy lost due to Mie scattering, \( \beta(\lambda)_{ns} \) is the fractional amount of energy lost due to non-selective scattering, \( \beta(\lambda)_{ext} \) is the summation of all scattering and absorption coefficients, and \( z \) is the path length of propagating beam.

In the thermal region of the spectrum (8-14 \( \mu m \)) scattering is negligible and the calculation for total transmission is reduced to the expression shown below in Equation A.12 under clear sky conditions.

\[ \tau(\lambda) = e^{-\beta(\lambda)_{\alpha}z} \]  
\hspace{1cm} \text{(A.12)}

This simplified relationship demonstrates that the atmospheric transmission in the thermal region under clear sky conditions is exclusively dependent on atmospheric absorption. This assumption breaks down when the atmosphere is hazy, filled with aerosols, or contains lots of moisture. Under those conditions a more rigorous atmospheric modeling tool, like MODTRAN, would be the appropriate approach to determine the atmospheric transmission [18].

A.7 Sensor-reaching radiance

In order to calculate the total sensor-reaching radiance, all potential self-emissive sources in the scene need to be accounted for, as well as the impact of atmospheric constituents.
on the detected radiance. Based on previous discussion, the radiance leaving the target is approximated using Planck’s equation and the material’s emissivity. This signal combines with the upwelled, downwelled, and background radiance terms (described in Section A.5) to form the observed sensor-reaching radiance. Mathematically, this combination is expressed below in Equation A.13.

\[ L_\lambda = (\epsilon_\lambda L_{BB} + F(1 - \epsilon_\lambda)L_d + (1 - F)(1 - \epsilon_\lambda)L_b)\tau_\lambda + L_u \]  

(A.13)

<table>
<thead>
<tr>
<th>$\epsilon_\lambda$</th>
<th>target spectral emissivity</th>
</tr>
</thead>
<tbody>
<tr>
<td>$L_{BB}$</td>
<td>total radiance emitted by blackbody at temperature $T$</td>
</tr>
<tr>
<td>$F$</td>
<td>fraction of above hemisphere observable by the target</td>
</tr>
<tr>
<td>$1 - \epsilon_\lambda$</td>
<td>target spectral reflectivity</td>
</tr>
<tr>
<td>$\tau_\lambda$</td>
<td>total atmospheric spectral transmission</td>
</tr>
<tr>
<td>$L_d$</td>
<td>downwelling self emitted radiance from sky dome</td>
</tr>
<tr>
<td>$L_b$</td>
<td>background self emitted radiance falling on target</td>
</tr>
<tr>
<td>$L_u$</td>
<td>upwelling self emitted radiance from atmosphere</td>
</tr>
</tbody>
</table>

Table A.3: Sensor-reaching radiance equation variables

Note, the $1 - \epsilon_\lambda$ term is equivalent to the target’s reflectivity as developed in Section A.4 via Kirchoff’s Law. In Equation A.13 the variables represent the following: $\epsilon_\lambda$ is the target spectral emissivity, $L_{BB}$ is the total radiance emitted by blackbody at temperature $T$, $F$ is the fraction of the sky observable by the target, $1 - \epsilon_\lambda$ is the target spectral reflectivity, $\tau_\lambda$ is the total atmospheric spectral transmission, $L_d$ is the downwelling self-emitted radiance from sky dome, $L_b$ is the background self-emitted radiance falling on target, and $L_u$ is the upwelling self-emitted radiance from atmosphere.
Appendix B

Temperature retrieval methods

As discussed previously in Section A, radiance signals detected at the sensor are a combination of several sources within and outside of the sensor’s field-of-view. Buried inside the observed radiant energy is information regarding a target’s temperature. In order to validate any sensor-reaching radiance signal, a calibration method needs to be implemented which translates the sensor-acquired digital counts into sensor-reaching radiance and ultimately the material’s apparent or absolute temperature. Calibration methods will either use in-scene targets consisting of known material characteristics or sensor-mounted blackbody calibration targets. Any method executed must take into account atmospheric contributions and material emissivity effects in order to derive an absolute material temperature.

B.1 Atmospheric compensation

Atmospheric compensation methods remove the effects of the atmospheric on an acquired radiance image. A modification is performed on the radiance signal observed at the sensor to compensate for both atmospheric transmission losses and upwelling radiance contributions. Unless additional steps are taken to account for surface emissivity and material surface properties, atmospheric compensation will only deliver apparent surface temperature instead of absolute temperature (i.e the temperature associated with the surface leaving radiance).
B.2 Correlation with ground-based measurements

The simplest method to derive absolute temperature from a radiance image is through the correlation of remotely observed data with ground-based measurements. While the resulting calibration is conceptually easy, this approach is labor intensive and prone to error due to the difficulty of obtaining simultaneous overhead and ground measurements without significantly interfering with the system being measured. This method does not make an attempt to perform any sort of atmospheric compensation, but instead aims to derive a functional relationship between the remotely sensed radiance and the parameter of interest, in this case temperature. This functional relationship will take the form shown in Equation B.1, where $Y$ is the parameter of interest and $DC_1\ldots DC_N$ are observed values at the different spectral bands of the sensor.

\[
Y = f(DC_1, DC_2, \ldots, DC_N) \quad (B.1)
\]

Generally, when there is no knowledge of the relationship between the parameter and the observed signal, a large amount of data is required to produce a functional regression form which successfully minimizes the residual error for the function and maintains a robust, constrained solution. The required amount of collected ground-based data points would be extremely large and need to coincide with image acquisition. These requirements make the task of successfully collecting enough data daunting. However, if given some knowledge about the environment and parameters being observed, an assumption can be made that the relationship between the observed signal and the parameter of interest is linear, a simple linear regression can be formed.

When this simpler approach is applied, several basic malleable equations fall out of the process. For temperature retrieval, the observed digital counts are first regressed against radiance observed from calibration targets. Assuming that the calibration targets emitted radiance values bracket the observed scene temperatures adequately, the resulting regression coefficients ($m_R$ and $b_R$ in Equation B.2) are used to translate a raw digital count image into a radiance image. The choice of adequate targets to bracket a scene thermally is significant in that it insures that your linear assumption between radiance and temperature will hold across the range of temperatures being observed.

\[
DC = f(L) = m_R L + b_R \quad (B.2)
\]
Next ground-based temperature measurements are correlated with their corresponding image-based radiance values to perform a second regression. The resulting coefficients ($m_T$ and $b_T$ in Equation B.3) from the second regression are then used to translate the radiance image into apparent ground temperatures.

$$L = f(T) = m_T T + b_T \quad (B.3)$$

Equation B.2 and B.3 can be combined to form the overall functional relationship between the observed digital count at the sensor and the ground temperature, shown in Equation B.4. Until further steps are made to take into account surface emissivity effects, the ground temperatures will not be absolute.

$$DC = m_R (m_T T + b_T) + b_R \quad (B.4)$$

A drawback to this approach is the lack of flexibility in the derived solutions. One is essentially avoiding performing a true atmospheric compensation technique by assuming the atmospheric affects can be accounted for by a linear regression. Because of this assumption, the derived linear coefficients will be unique to the collection conditions and environment and will only be valid for the single data set. Additionally, the assumption is made that not only is the detector’s response linear with radiance, but the observed radiance values are linearly related to temperature. Given the highly non-linear behavior of the Planck equation, care needs to be take that within the spectral and thermal ranges of interest, the Planck equation does behave linearly. This behavior can be demonstrated by calculating the Plank equation using the ranges desired as well as verifying the relationship via any data collected.

### B.3 Ground-based temperature measurement

The following approach aims calculate the atmospheric calibration values, $\tau$ and $L_u$, when calibrating a thermal scene using collected ground truth. The difference between this method and the previous is this technique allows for more flexibility in the number of in-scene targets, assuming the emissivities of the varying targets are well characterized. Essentially, the radiance values observed by a calibrated sensor can be regressed against the measured emitted radiance from ground targets at some temperature. By choosing
targets that behave as blackbodies, the governing Equation A.13, is reduced to Equation B.5 where \( L \) is the sensor-observed radiance, \( \tau \) is the atmospheric transmission, and \( L_u \) is the upwelling radiance.

\[
L = L_T \tau + L_u \tag{B.5}
\]

When applying this simplified equation to this ground truth technique, the atmospheric transmission and upwelling radiance are captured in the gain \((m = \tau)\) and bias \((b = L_u)\) calculated by the regression. In order to apply the calculated coefficients to other materials in the scene (assuming they are gray bodies with known emissivities), the slope and intercept of the regression absorb the residual error introduced by the target emissivity and downwelled radiance, shown in Equation B.6 where \( L \) is the sensor-observed radiance, \( \epsilon \) is the target emissivity, \( \tau \) is the atmospheric transmission, \( L_d \) is the downwelling radiance, and \( L_u \) is the upwelling radiance.

\[
L = L_T \epsilon \tau + \tau r L_d + L_u \tag{B.6}
\]

Equation B.6 can be rearranged into Equation B.7 and the regression relationship is evident. The quantity \( \epsilon \tau \) is contained in the gain and the quantity \( \tau r L_d + L_u \) is contained in the bias.

\[
L = (\epsilon \tau) L_T + (\tau r L_d + L_u) \tag{B.7}
\]
Appendix C

Direct temperature retrieval

When measuring temperature, the true challenge is to derive a material’s absolute temperature without influencing the temperature with the measurement itself. There are two different methods of measuring temperature: contact and non-contact measurements. Contact measurements rely on making physical contact with the target of interest and is usually accomplished with a device such as a thermistor or thermocouple. Non-contact measurements rely on the same principles that allow emitted radiance to be observed from either aerial or satellite platforms. Non-contact devices, such as infrared radiometers, are simple single detector instruments that observe the radiance being emitted by the target in the longwave infrared region of the spectrum. The measurement is usually at close range to minimize atmospheric effects in the observed signal.

C.1 Contact vs. non-contact methods

Any type of contact measurement inherently interacts with the material surface being measured and induces an energy flow across the interface between the material and the instrument. Essentially, a contact device will measure its own temperature once it has reached thermal equilibrium with the target. By being in contact with the material, the contributors of error in measurement are limited to the material and the instrument.

When observing the temperature of a material via non-contact methods, there is no influence of the instrument on the material, but the phenomenon being observed is different. As discussed in Section A, energy being emitted by a target is related to the target’s temperature through Planck’s equation and is not directly measurable by remote
techniques. Therefore, additional processing must be performed on non-contact measurements to derive the target’s temperature. Many commercially available instruments (e.g. Heitronics KT19.82 and Omega OS36 radiometers) will perform this task for the user, however, many times the process implemented is somewhat of a black box with the user having little insight into how the target emissivity and the spectral response of the detector are being handled.

C.2 Skin temperature effects

The surface temperature at an air-water interface is not a trivial measurement. Significant temperature measurement differences arise when an observation is made using a contact device at some depth in the water column below the surface (referred to as $T_{\text{bulk}}$) and when the surface is observed by a non-contact radiometer (referred to as $T_{\text{skin}}$). The reason for the difference is directly related to the energy transfer mechanisms at work at the interface. In the top 10-100$\mu$m of the interface the energy fluxes are dominated by molecular conduction. A positive temperature gradient with depth, referred to as the “cool skin effect”, across the molecular layer is required to maintain the net heat flux in the direction from the water to the air.

The temperature-depth profile of a given water column will be strongly dependent on the amount of solar flux falling on the surface and the wind speed. As wind blows across a body of water the surface is cooled by the convective forces. As wind speed increases it will dominate any influence of solar flux and lead to a more thermally mixed near-surface layer. In contrast, light to no wind conditions allow the solar heating to dominate and induce a large thermal stratification through the water column as a function of depth. Figure C.1 shows the difference between idealized temperature-depth profiles for different wind conditions.

Generally speaking, the net flux will always move in the direction from the water to the air because of net radiation losses of a warmer water body to a colder sky. It is possible, however, that with high solar loading on an interface and minimal wind (less than 1 m/s), the cool skin effect can be overcome to produce either no skin effect or a warm skin effect.
C.2. Skin temperature effects

Figure C.1: Idealized temperature profiles of the near-surface layer of a water body. Blue line represents the temperature profile of a body water during the day in low wind conditions. The purple, dotted line represents the temperature profile of the same body of water during the night or day in strong wind conditions.
Appendix D

Error propagation

No measurement device or experimentalist is perfect and will inject an error into any measured quantity. In order to quantify the error introduced by a particular individual measurement, the total standard error is calculated. The total standard error in a measurement, \( \sigma_m \), is determined by taking the root sum square of the accuracy, \( \sigma_i \), and precision, \( \sigma_p \), metrics for the measurement. The precision of a measurement describes its repeatability and the accuracy of a measurement describes how closely an instrument can match a value determined to be truth. The calculation for the total standard error is shown below in Equation D.1 [22].

\[
\sigma_m = \sqrt{\sigma_p^2 + \sigma_i^2} \quad \text{(D.1)}
\]

For a particular process, where the desired parameter can be described using a general governing equation, a simple expression (derived by Beers [5]) can be used to derive contributions for each single source of error to the total measurement error. The dependent variable \( Y \) is calculated using a functional form with \( N \) independent variables \( X_i \) as inputs, shown in Equation D.2.

\[
Y = f(X_1, X_2, \ldots, X_N) \quad \text{(D.2)}
\]

The standard error in \( Y \), \( \sigma_Y \), is expressed as the following relationship of individual input variable errors, \( \sigma_{X_i} \).
\[
\sigma_Y = \left[ \left( \frac{\delta Y}{\delta X_1} \sigma_{X_1} \right)^2 + \left( \frac{\delta Y}{\delta X_2} \sigma_{X_2} \right)^2 + \cdots + \left( \frac{\delta Y}{\delta X_N} \sigma_{X_N} \right)^2 \right]^{\frac{1}{2}} \tag{D.3}
\]

When the input variables, \(X_i\), are correlated, however, the total error calculation must take into account the dependency between the various input parameters. This compensation is accomplished by including, \(\rho_{xy}\), the correlation coefficient between each pair of variables [5]. Below, the formula shown in Equation D.4 is calculating the variance, \(\sigma_Y^2\), in the function \(Y\) and is related to the standard error through the square root.

\[
\sigma_Y^2 = \left( \frac{\delta Y}{\delta X_1} \sigma_{X_1} \right)^2 + \left( \frac{\delta Y}{\delta X_2} \sigma_{X_2} \right)^2 + \cdots + \left( \frac{\delta Y}{\delta X_N} \sigma_{X_N} \right)^2 \\
+ \sum_{i,j} 2\rho_{ij} \left( \frac{\delta Y}{\delta X_i} \right) \left( \frac{\delta Y}{\delta X_j} \right) \sigma_{X_i} \sigma_{X_j} \tag{D.4}
\]
Appendix E

WASP sensor calibration

E.1 Digital counts to radiance

Raw data from both the LWIR and MWIR detectors on WASP are written out as 14-bit ITTVIS ENVI format images. In order to minimize projection effects and preserve the radiometric integrity of the raw data contained in these images, the conversion from digital number to radiance units is performed prior to the georeferencing process. Additionally, due to significant non-uniformity across the LWIR focal plane and variable readout gain visible in the MWIR focal plane, each image is calibrated to sensor-reaching radiance on a pixel-by-pixel basis.

For most aerial collections, blackbody imagery is collected at the beginning and end of each flight line. Each reference source is driven to either a hot or cold set point and moved to fill the field of view for both sensors simultaneously. The hot and cold set points are chosen to adequately bracket the scene’s thermal content of interest. After the first reference source imagery collection (at either a hot or cold set point), the blackbodies are driven to the opposite end of the thermal range during the imagery acquisition over the scene, and then re-introduced into the field-of-view while the aircraft is making a turn. Figure E.1 is a conceptual example for how this type of imagery collection scheme is implemented.

Each collection of reference source imagery is temporally averaged to reduce noise during collection. Prior to any radiometric calculation, an analysis is performed to identify and interpolate any dead pixels on the detector’s focal plane. It is assumed that 0.05% of the total number of pixels are dead or misbehaving. Using the histogram statistics
for a given average blackbody image, the top 0.025\% and bottom 0.025\% of pixels are designated as dead pixels and are filled with a bi-linear interpolation of neighboring pixel values.

Given the known spectral emissivity of the blackbody material, \( \epsilon(\lambda) \), the spectral response of each of the WASP thermal cameras, \( R(\lambda) \), and the measured temperature of the blackbody during image acquisition, the total radiance emitted by the reference source and detected by the WASP detector, \( L \), is determined by integrating the Planck blackbody equation over the spectral range of the WASP detector. This calculation is performed for both the hot- and cold-averaged blackbody images.

\[
L_{\text{hot}} = \int_{\lambda} L_{BB,T_{\text{HOT}}}(\lambda)R(\lambda)\epsilon(\lambda)
\]

\[
L_{\text{cold}} = \int_{\lambda} L_{BB,T_{\text{COLD}}}(\lambda)R(\lambda)\epsilon(\lambda)
\]

Because the WASP LWIR detector response is linear with radiance, for each flight, the
digital count values at every pixel for both the hot- and cold-averaged blackbody images is related to the calculated total emitted radiance from the reference sources using a linear regression model. Due to the fact that the set points for the reference sources were chosen based on the thermal content of target scene, the gain ($m_R$) and bias ($b_R$) calculated for each pixel will generate an apparent sensor reaching radiance value for any given digital count acquired while imaging the scene (Figure E.2).

$$L_{app,i,j} = m_{R,i,j} DC_{i,j} + b_{R,i,j}$$

(E.3)

The end result of this process are gain and bias masks for each flight line. These masks are applied to each image acquired during a flight line to convert the raw data into apparent sensor reaching radiance. Following this conversion, image tiles are geo-referenced and stitched together to create a single image mosaic encompassing the entire scene of interest.

### E.2 Radiance to temperature

In order to compensate for atmospheric effects and correlate to apparent ground temperature, georeferenced ground truth measurements of the target (water surface) were collected. Both skin and bulk temperature measurements of the water were recorded, however, the skin temperature measurements were used for calibration efforts. The choice to use only skin temperature measurements was made because a skin temperature model would have needed to be implemented to convert bulk measurements to surface measurements. None of the models researched adequately handled the highly dynamic environment observed on the cooling pond.

Using the map coordinates associated with each measurement, the corresponding calculated radiance value was extracted from the georeferenced, sensor-reaching radiance mosaic. It was assumed that over the range of temperatures investigated and using the WASP system, the relationship between sensor-reaching radiance and target temperature was linear. To confirm that not only a linear relationship was maintained over the entire thermal range, but also over the smaller range of actual target temperatures, the total theoretical emitted radiance for a water target at a given temperature, as observed by the WASP LWIR detector, was calculated. The relationship between temperature and radiance is demonstrated below in Figure E.4. It should be noted that when fit with a linear
CHAPTER E. WASP sensor calibration

Figure E.2: Digital count to radiance conversion depicting the linear relationship assumed between the digital counts recorded at each pixel and the apparent radiance emitted from the blackbody at each temperature set point.

\[ L_i = m_r [DC_i] + b_r \]

\[ m_r = \frac{\Delta L}{\Delta DC} \]

model, the gain and bias remain relatively unchanged for both the full thermal range and the subset.
E.2. Radiance to temperature

Figure E.3: Relationship between radiance and temperature for entire range possible scene temperatures as well as a fitted linear model.

Figure E.4: Relationship between radiance and temperature for subset of thermal range of possible scene temperatures as well as a fitted linear model.
The calibrated, ground-collected, skin temperature measurements were then compared to the corresponding sensor-reaching radiance values. All radiance values are corrected for the water target emissivity ($\epsilon_{\text{water}} = 0.987$) as shown in Equation E.4.

$$L_{\text{obs}} = \frac{L_{\text{app}}}{\epsilon_{\text{water}}} \quad (E.4)$$

Based on the aforementioned linear relationship assumption, a linear model was fit to the radiance-temperature data pairs, producing a gain ($m_T$) and bias ($b_T$) that is used to convert from sensor-reaching apparent radiance to the ground temperature of water at a given location. This relationship is shown in Equation E.5.

$$T_{\text{obs}} = m_T L_{\text{obs}} + b_T \quad (E.5)$$

The gain and bias are single values that are only valid for the particular data campaign they were collected for and the corresponding thermal imagery. The relationship between the radiance temperature pairs is depicted in Figure E.5. It is important to note that all derived ground temperatures are only valid for water.

![Figure E.5: Graphic depicting apparent sensor-reaching radiance to temperature conversion.](image)

Using both Equation E.3 and Equation E.5, a governing equation can be produced for the entire digital count to ground observed temperature conversion process (Equation E.6).
\[ L_{obs_{i,j}} = m_{R_{i,j}} DC_{i,j} + b_{R_{i,j}} \]
\[ T_{obs} = m_T L_{obs} + b_T \]
\[ T_{obs_{i,j}} = m_T [m_{R_{i,j}} DC_{i,j} + b_{R_{i,j}}] + b_T \] (E.6)
Appendix F

Ground instrument calibration

An additional step was taken to calibrate the instrument-acquired measurements to actual temperature using instrument calibration data acquired in the field. A Heitronics KT19.82 radiometer and Omega OS36 radiometer were used to collect skin temperature measurements for the results presented. Calibration points were collected using a portable Omega blackbody calibration source that was driven to temperatures that thermally bracketed the water targets to be measured. A linear model, Equation F.1, was used to generate a gain ($m_C$) and bias ($b_C$) to be used to convert observed temperatures to absolute temperatures.

$$T_{obs} = m_C T_{abs} + b_C$$  \hspace{1cm} (F.1)

Field instruments (Heitronics KT19.82 radiometer and Omega OS36 radiometer) were calibrated in the field using a portable blackbody radiative source. Shown in Figure F.1 are the calibration curves resulting from the in-field calibration. All on-board instrumentation built into the buoys was calibrated, pre-deployment, using controlled thermal conditions in the laboratory.
Figure F.1: Calibration plot for skin temperature retrieval instruments generated by data collected in the field using a blackbody thermal source.
Appendix G

WASP sensor sensitivity analysis

The entire process of conversion from digital number to absolute ground temperature can be described using a single governing equation (Equation G.1) based on all the linear regression models used to determine the final temperature. The process to develop Equation G.1 is shown below. Because the conversion from digital count to sensor-reaching radiance was performed on a pixel-by-pixel basis, a temperature is derived for every pixel using a unique gain ($m_R$) and bias ($b_R$). All variables are considered independent and uncorrelated.

\[
\begin{align*}
T_{obs_{i,j}} &= m_T [m_{R_{i,j}} DC_i + b_{R_{i,j}}] + b_T \\
T_{obs} &= m_C T_{abs} + b_C \\
m_C T_{abs} + b_C &= m_T [m_R DC + b_R] + b_T \\
T_{abs_{i,j}} &= \frac{m_TM_{R_{i,j}} DC_{i,j} + m_T b_{R_{i,j}} + b_T - b_C}{m_C} 
\end{align*}
\] (G.1)

Applying Equation D.4 to Equation G.1 produces the formula (Equation G.2) for determining the variance in a produced temperature map of the water on a pixel-by-pixel basis. The variables are independent and uncorrelated so the last term in Equation D.4 can be ignored. The final error measure will have the same units as the governing equation (temperature, K). It is important to note that because the temperature is calculated on a pixel-by-pixel basis, an error will be calculated for every pixel, generating a corresponding error map for each temperature map.
\[
\sigma_{T_{\text{abs},i,j}}^2 = \left( \frac{\delta T_{\text{abs},i,j}}{\delta m_C} \right)^2 \sigma_{m_C}^2 + \left( \frac{\delta T_{\text{abs},i,j}}{\delta m_T} \right)^2 \sigma_{m_T}^2 + \left( \frac{\delta T_{\text{abs},i,j}}{\delta m_{R_{i,j}}} \right)^2 \sigma_{m_{R_{i,j}}}^2 
+ \left( \frac{\delta T_{\text{abs},i,j}}{\delta D_{C_{i,j}}} \right)^2 \sigma_{D_{C_{i,j}}}^2 + \left( \frac{\delta T_{\text{abs},i,j}}{\delta b_{R_{i,j}}} \right)^2 \sigma_{b_{R_{i,j}}}^2 + \left( \frac{\delta T_{\text{abs},i,j}}{\delta b_T} \right)^2 \sigma_{b_T}^2 
+ \left( \frac{\delta T_{\text{abs},i,j}}{\delta b_{C_{i,j}}} \right)^2 \sigma_{b_{C_{i,j}}}^2 
\tag{G.2}
\]

Each of the error terms in Equation G.2 are calculated from various sources and methods. A method called bootstrapping is used to calculate the linear regression used in both the radiance to temperature conversion (Equation E.5) and the ground instrument calibration (Equation F.1). Bootstrapping is a method of calculating the properties of a statistical estimator (e.g. variance) by measuring those properties from approximating distributions [8]. The approximating distributions are created from replicating the sample distribution shape’s through a sampling with replacement technique. Each one of the approximate distributions are referred to as a bootstrap sample. The estimator of interest is calculated for each bootstrap sample. Additionally, with this increase data the variance of the estimators can be determined. This method is used to determine the variance in both the gain and bias of the calculation. This method was implemented using an existing routine, written in IDL. This function calculated the values for \( \sigma_{m_T}^2, \sigma_{b_T}^2, \sigma_{m_C}^2 \) and \( \sigma_{b_C}^2 \).

Both \( \sigma_{m_{R_{i,j}}}^2 \) and \( \sigma_{b_{R_{i,j}}}^2 \) have values of 0 because of how the variables whose variance they represent was calculated. For each pixel a single hot value and single cold value are used to produce the pixel’s gain and bias for is conversion to radiance space. Because this linear fit is based on only two points, instead of a series, there is no error in the calculation. In order to calculate the variance in the raw WASP image, \( \sigma_{D_{C_{i,j}}} \), the variance is calculated for each pixel. During the averaging of the blackbody images for calibration, the standard deviation is calculated for each pixel. Figure G.1 illustrates this process.
Figure G.1: Graphic illustrating the calculation of the variance terms for each pixel. The variances are calculated for each pixel using the average of each series of blackbody images. These calculated variances are then regressed against their corresponding digital counts to determine the variance gain and bias masks.

These standard deviation images are calculated for both the hot and cold averaging processes. Using the average blackbody images as independent variables, a linear regression is calculated between the digital counts in the images and their corresponding standard deviations. The results of this regression are a gain and bias to calculate a standard deviation image from a raw WASP image. Because the blackbody averaging is accomplished with every flight line, this technique has the ability to absorb any error from non-uniformities that may arise after prolonged operation.
Appendix H

Imagery Calibration

Two sets of data, that were calibrated using the aforementioned process (Sections 5.1.4 and G). Both data sets were acquired at night, however the water surface temperature data acquired during the 11 February 2010 collect was measured using the Heitronics KT19.82 radiometer while the data acquired during the 4 March 2010 collect was measured using the Omega OS36 radiometer. Each data set was processed to generate both temperature maps (Figure H.1 and Figure H.2) and error maps. Derived temperatures are only valid for water and each pixel’s error is designated by the corresponding per-pixel error map.
CHAPTER H. Imagery Calibration

Figure H.1: Calibrated temperature map for the 11 February 2010 night collect created using the implemented calibration technique. Temperature is reported in Celsius.

Figure H.2: Calibrated temperature map for the 4 March 2010 night collect created using the implemented calibration technique. Temperature is reported in Celsius.
Table H.1: Statistics from uniform area of generated error maps

<table>
<thead>
<tr>
<th>Date</th>
<th>Mean $\sigma_{T_{cal}}$</th>
<th>Standard deviation in $\sigma_{T_{cal}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>02/11/10</td>
<td>0.94</td>
<td>0.02</td>
</tr>
<tr>
<td>03/04/10</td>
<td>0.50</td>
<td>0.01</td>
</tr>
</tbody>
</table>

Shown in Table ?? are the mean-calculated standard deviations for a section of approximately uniform temperature water as well as the amount of variation about that calculated standard deviation. The data collected during the 4 March 2010 collection has a distinctly better accuracy which is directly attributable to quality of ground truth collected and the atmospheric homogeneity.

Figure H.3: Radiance to ground temperature calibration model for the 11 February 2010 night collect. Each point represents a geo-located temperature measurement and its corresponding sensor-reaching radiance.

Figure H.3 and Figure H.4 show the collected surface temperatures for each day plotted against the corresponding radiance values extracted from the sensor-reaching radiance mosaics. It can be seen that surface temperature data collected during the 4 March 2010 collect had a stronger linear relationship to the sensor-reaching radiance than the collected
data from the 11 February 2010 collect. Additionally, the measurements made are more evenly distributed throughout the range of possible temperatures.

![Figure H.4: Radiance to ground temperature calibration model for the 4 March 2010 night collects.](image)

Figure H.4: Radiance to ground temperature calibration model for the 4 March 2010 night collects. Each point represents a geo-located temperature measurement and its corresponding sensor-reaching radiance.

The notable variation between the two days’ ground data can be attributed to the strength of the linear fit generated for the translation between sensor-reaching radiance and ground observed temperature. The ground truth collected during the 4 March 2010 collect has a distinctly stronger linear relationship than the data collected for 11 February 2010 collect. As the linear fit between sensor-reaching radiance and ground observed temperature weakens, the error terms attributed to the linear model, $\sigma_{mT}$ and $\sigma_{bT}$, grow and as a result increase the magnitude of the derived standard deviation.

This strong dependence on well fit ground truth data can be expected due to the linear assumption made regarding the relationship between radiance and temperature for a given target. Under that assumption, the atmospheric affects are believed to have a uniform, homogeneous affect on the emitted radiance from a given target; an affect that can accounted for by a simple gain and bias. As soon as the atmosphere between the WASP cameras and water surface becomes more spatially variant, this assumption deteriorates.

For the particular environment observed for this research, the linear assumption does not
consistently hold true. The body of water being observed has a highly varying thermal structure that produces a localized boundary layer micro-climate. In addition, the lake is accompanied by cooling towers, positioned on shore, which introduce a significant amount of moisture into the atmosphere. The presence of these phenomena are directly influencing the atmospheric make-up of the air column above the pond and can lead to a distinctly complicated radiometric environment.
Appendix I

Autonomous in-situ measurements

I.1 Winter 2008-2009

The data campaign for the winter of 2008-2009 was segmented into two major branches: the aerial data collection and the ground truth collection. While the aerial data collection has a single node of collection, the ground truth collection was a multi-faceted venture involving both manual and automatic measurements of water surface temperature, ice thickness, and weather conditions. Five buoys were deployed in the cooling lake of the Midland Cogeneration Venture (MCV), while a weather station was constructed on the shore. The buoys, as well as the weather station, took automatic measurements of various parameters of interest and transmitted the data on daily intervals via cellular modems. In addition to the constant monitoring provided by the buoys, manual, high-density surface temperature measurements and ice thickness measurements were taken immediately following imagery collects. Flights were attempted on a weekly interval, as weather permitted. Five identical buoys were deployed within the lake. Each buoy was equipped with a Campbell Scientific CR1000 datalogger to perform all data acquisition as well as control all telecommunications and GPS monitoring. The internal components of a constructed buoy is shown below in Figure I.1(a).

The CR1000 monitored two different deployments of thermocouples used to measure the temperature profile of the water and ice thickness. The temperature profile thermocouples were attached to the mooring chain at one-foot increments and were queried by the datalogger at 30-second intervals. Every five minutes an average temperature value at each thermocouple was calculated and recorded. The second set of thermocouples was
CHAPTER I. Autonomous in-situ measurements

(a) Internal buoy components

(b) Ice thickness probe

Figure I.1: Inside of buoy containing datalogger, multiplexors, GPS units, battery, and cellular modem shown in I.1(a). Ice thickness probe (long, black pole) mounted on side of buoy shown in I.1(b)

attached to a metal pole, at approximately 1-inch increments, beginning at the base of the buoy. These high-density thermocouples measured the temperature of either water or ice every 30 seconds with an average temperature recorded every five minutes. In addition to the main measurement systems on the buoys, twelve Tidbit dataloggers were attached to the chain of each buoy to serve as a backup temperature measurement system. Notable problems from the 2008-2009 winter season included: power failures, communication failures, challenging accessibility issues for maintenance, lack of insight into the actual 3-axis positioning of a given buoy, and lack of a backup system for collected data.

I.2 Winter 2009-2010

Following the challenges of the 2008-2009 collection efforts, a massive overhaul and re-engineering of the deployable buoy monitoring systems was completed. While the basic construction of the buoys remained unchanged, in order to combat the system failures mentioned in Section 5.1.2, modifications were made to the buoy design as well as the controlling software. A comparison between the two design approaches are shown in Figure J.1 The internal components of a newly designed and constructed buoy is shown above in Figure I.4(a). A compact flash memory card was installed on each buoy to serve as a backup should there be a communication failure. In addition, an accelerometer was added to report on the 3-axis orientation of a given buoy. This additional information
I.2. Winter 2009-2010

(a) Buoy floating in open water
(b) Buoy embedded in ice and snow

Figure I.2: Buoys in both open water and iced conditions while deployed in the MCV cooling pond during the 2008-2009 winter collection season.

provided insight into the angle at which a buoy was oriented should it become frozen in the ice. The angle at which a buoy freezes directly affects the position of any thermocouple on the ice profiler and, by association, the derived location of an ice/water boundary.

In addition to a dynamic mooring chain assembly, the float size and buoyancy rating was increased from 200 lbs buoyancy to 700 lbs buoyancy to increase the platform’s resistance to freezing at non-horizontal angles. To address power failure issues, a 12-watt solar panel was mounted to the top of the environmental enclosure and the battery location was moved to an external mount point, shown in Figure I.4(b). Each weatherproof box containing all electrical components was permanently sealed shut to prevent any water seepage. In case of software failures mid-winter, the capability to manually turn on the modem was added via a physical toggle switch on the antenna. This added functionality allowed the software and firmware on the datalogger to be modified via a wireless internet connection and alleviated the need to access the datalogger while deployed. Figure I.5 shows the buoys successfully deployed in the MCV cooling pond [3].
CHAPTER I. AUTONOMOUS IN-SITU MEASUREMENTS

Figure I.3: Comparison of buoy design differences between the 2008-2009 and 2009-2010 winter collection seasons.
I.2. Winter 2009-2010

(a) Internal buoy components  
(b) Externally mounted solar panel

Figure I.4: Newly designed buoys in the RIT laboratory

Figure I.5: Newly designed buoys afloat in MCV cooling pond
Appendix J

Ice thickness probes

Five identical buoys were deployed within the lake. Each buoy is equipped with a Campbell Scientific CR1000 datalogger to perform all data acquisition as well as control all telecommunications and GPS monitoring. The internal components of a constructed buoy is shown below in figure J.1(a).

![Internal buoy components](image)

![Ice thickness probe](image)

Figure J.1: Inside of buoy containing datalogger, multiplexors, GPS units, battery, and cellular modem shown in J.1(a). Ice thickness probe (long, black pole) mounted on side of buoy shown in J.1(b).

The CR1000 data loggers built into the buoys monitored two different deployments of thermocouples used to measure the temperature profile of the water and ice thickness. The temperature profile thermocouples were attached to the mooring chain at one-foot
increments and were queried by the datalogger at 30-second intervals. Every five minutes
an average temperature value at each thermocouple was calculated and recorded. The
second set of thermocouples was attached to a metal pole, at approximately 1-inch incre-
ments, beginning at the base of the buoy. These high-density thermocouples measured the
temperature of either water or ice every 30 seconds with an average temperature recorded
every five minutes. In addition to the main measurement systems on the buoys, twelve
Tidbit dataloggers are attached to the chain of each buoy to serve as a backup temperature
measurement system.

Attached to each deployed buoy was an ice thickness probe consisting of 46 evenly-
spaced thermocouples. Temperatures were recorded at each thermocouple and associated
to their position relative to the top of the probe. Although designed so the first ther-
mcouple would be approximately at water level, the actual location of this data point
relative to the ice surface varied based on freezing conditions. The overall thickness of ice
(if present) was determined by assuming any thermocouples registering temperatures of
0° or below were imbedded in ice. It was understood that this method of ice thickness
determination would have larger errors during the freeze and melt periods. In figure J.1(b)
the ice-thickness probe can be seen attached to the side of the buoy.

In order to determine the validity of this method ice holes were periodically drilled
in close proximity to buoys that had been frozen in ice and the true thickness of the ice
was measured. An ice thickness value is obtained from recorded buoy data through visual
inspection of the thermocouple output. Below in figure J.2 are four examples of plotted ice
probe output. Figures J.2(a) and J.2(b) show the depth versus temperature recorded for
the same buoy, Shiva, on two different days. Figures J.2(c) and J.2(d) shows the analogous
information for a second buoy, Surya. It is important to note that on buoy Surya there
was an inoperable thermocouple at the 17 inch mark.

From visual inspection it is assumed that the first visible inflection point marks the first
ice (or snow) interface with air. The second inflection point is designated the ice/water
interface. The difference between these two points is calculated as the ice thickness. The
images in figure J.2 show the inflection points occurring at values higher than 0°. Due
to manually observed and measured ice thickness values it is known that these plots do
correspond to probes imbedded in ice. It is possible that the skew in temperature is a
calibration issue with the thermocouples. In table J.1 the comparison between manually
and automatically measured ice thickness is shown.
Table J.1: Measured ice thickness versus buoy-recorded ice thickness

<table>
<thead>
<tr>
<th>Buoy</th>
<th>Measured by hand [in]</th>
<th>Recorded by buoy [in]</th>
<th>Snow depth [in]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shiva 1/14</td>
<td>7.5</td>
<td>12</td>
<td>3</td>
</tr>
<tr>
<td>Shiva 2/3</td>
<td>11</td>
<td>16</td>
<td>0</td>
</tr>
<tr>
<td>Surya 1/14</td>
<td>9</td>
<td>9</td>
<td>4.5</td>
</tr>
<tr>
<td>Surya 2/3</td>
<td>13</td>
<td>8</td>
<td>2</td>
</tr>
</tbody>
</table>

There are several factors that could be responsible for the difference between manually measured and buoy observed ice thickness values. The buoy observed values are inferred from the thermocouple readouts at a specific time of the day. It has been observed that if the recorded thermocouple plots are viewed as a series of time-based frames in an animation the inflection points for the two surfaces are better defined and appear as almost stationary points around which the bordering temperatures fluctuate. Performing a time series analysis to extract the interface points might lead to more accurate total thickness measurements. The presence of snow could also influence the thermocouple readings, particularly in windy conditions. Finally, it is not guaranteed that the buoys will freeze parallel to the water surface. As a result of this misaligned freezing the assumed depth resolution of the ice probe is no longer valid and the angle of the imbedded probe would have to be known to accurately extract the true depths of each thermocouple.
Figure J.2: Plotted thermocouple output from ice thickness probe.
Appendix K

Skin temperature experiment

In an effort to investigate different water surface temperature measurement techniques for use on the ground during data campaigns, two different measurement techniques were investigated in the laboratory. The goal of the experiment was to determine which technique would resolve the skin temperature effect the best. An assumption was made that under the conditions created in the lab a skin temperature effect would manifests itself. The first method used a thermistor in contact with the water’s surface. The thermistor was mounted on a piece of styrofoam in an attempt to place the contact face of the thermistor right at the water’s surface, shown in Figure K.1(a).

![Thermistor/float apparatus measuring water surface](image)

(a) Thermistor/float apparatus measuring water surface

![Radiometer measuring surface temperature](image)

(b) Radiometer measuring surface temperature

Figure K.1: Experimental set up for skin temperature observations

The second used a radiometer to remotely sense the water skin temperature. The radiometer (Omega OS136) was held by hand approximately an inch from the water’s surface, shown in Figure K.1(b). A high-precision mercury thermometer was used to
measure the bulk water temperature. A temperature controlled water bath was used as the experiment environment and the measurements were carried out in an indoor lab over two days. There was an unintentional 7°C change in ambient temperature conditions in the lab over the two day period. The water temperature was varied from 4 – 30°C in 2°C increments. At each temperature increment the radiometer recorded three surface temperatures while the thermistor measured the surface and bulk temperature, three times each. The thermistor measured the bulk temperature by being submerged into the water bath. Each measurement was repeated for turbulent and calm water conditions.

Figure K.2: Resulting temperature differentials in turbulent water conditions. The red line represents the difference between the surface temperature observed by the radiometer and the measured bulk water temperature. The green and purple lines indicate the differences between the thermistor and the two bulk measurements.
Figure K.3: Resulting temperature differentials in turbulent water conditions. The red line represents the difference between the surface temperature observed by the radiometer and the measured bulk water temperature. The green and purple lines indicate the differences between the thermistor and the two bulk measurements.

The results shown in Figure K.2 are the calculated temperature differences while measuring in turbulent water conditions. Because the difference in measurements were so small relative to the overall range of the experiment, the disparity between the types of measurement techniques are more easily resolvable by examining their temperature differentials. Each plotted line represented the difference between two measured quantities. For example, the red line is showing the difference between the temperature observed by the radiometer and the bulk temperature measured by the mercury thermometer. The positive temperature gradient between these two measurement series is the skin temperature effect. Comparatively, the green and purple lines are the results from the thermistor measurements. As one can see, the difference between the thermistor’s surface temperature measurement and the bulk temperature measurement (green line) are nearly nonexistent. Additionally, the difference between the two thermistor readings (purple line), one at the surface and one submerged, are essentially zero. These results indicates that weather the thermistor is at the surface or submerged, it is measuring the same quantity. Figure K.3
shows the same experimental results as Figure K.2, however these measurements were collected under calm water conditions. As evident by these results, the thermistor still fails to capture the skin temperature effect while the radiometer is successful.

The results from the skin temperature experiment demonstrate the thermistor technique’s inability to measure true water surface temperature. For future ground truth data campaigns, the radiometer approach was the preferred method for measurement collection.
Appendix L

On-board blackbody calibration

The blackbody sources were calibrated separately inside a walk-in cooler with thermal controls. The overall ambient temperature of the cooler was set at approximately \(5^\circ C\). Because this temperature could fluctuate \(\pm 10^\circ C\), depending on the external temperature, dew point, and how often the cooler was entered, the ambient temperature was constantly monitored. The blackbody source, which fills the LWIR detector’s field-of-view during flight, was placed on a table inside the cooler and driven from a temperature of \(40^\circ C\) to \(0^\circ C\) at \(5^\circ C\) increments. The surface temperature of the blackbody was measured using an Exergen infrared contact radiometer at five locations. The location layout and numbering scheme is depicted below in Figure L.1.

Figure L.1: Depiction of measurement locations on the surface of the LWIR blackbody source

The temperatures measured at the surface by the Exergen were compared to the temperature set by the WASP control software. The results were plotted to show the overall difference between the blackbody set point and the average of the five position measurements. The average plot is shown in Figure L.2 as well as the plotted difference between
the set point and average measured surface temperature. Because the difference between the set point and measured surface temperature is small compared to the range of temperatures measured, the actual difference is better visualized in Figure L.3.

Figure L.2: Average blackbody spatial temperature compared to set point. LWIR-t (red line) represents the set point line while LWIR (blue line) represents the measured surface temperature.
In addition, the measurements were compared separately, based on their spatial location, to the blackbody set point, to determine if there were any biases across the plane of the blackbody plate. Both plots are shown below in Figure L.4.
Overall the average bias fluctuates between $\pm 0.5^\circ C$, except for at the $0^\circ C$ point where the bias jumps to $1.5^\circ C$. This large jump is most likely due to the dew point being reached inside the cooler chamber and condensation forming on the blackbody surface. Spatially, all locations tend to trend in the same direction and exhibit the same behavior. It was concluded that spatially the blackbody plate is consistent and demonstrates no spatial dependency on radiance. In addition, the margin of error introduced by the blackbody process is approximately $\pm 0.5^\circ C$.
Appendix M

Differential Equations: Sensitivity Analysis

\[ T_{abs,i,j} = \frac{m_T m_{R,i,j} DC_{i,j} + m_T b_{R,i,j} + b_T - b_C}{m_C} \quad (M.1) \]

\[ \sigma_{T_{abs,i,j}}^2 = \left( \frac{\delta T_{abs,i,j}}{\delta m_C} \right)^2 \sigma_{m_C}^2 + \left( \frac{\delta T_{abs,i,j}}{\delta m_T} \right)^2 \sigma_{m_T}^2 + \left( \frac{\delta T_{abs,i,j}}{\delta m_{R,i,j}} \right)^2 \sigma_{m_{R,i,j}}^2 
+ \left( \frac{\delta T_{abs,i,j}}{\delta DC_{i,j}} \right)^2 \sigma_{DC_{i,j}}^2 + \left( \frac{\delta T_{abs,i,j}}{\delta b_{R,i,j}} \right)^2 \sigma_{b_{R,i,j}}^2 + \left( \frac{\delta T_{abs,i,j}}{\delta b_T} \right)^2 \sigma_{b_T}^2 
+ \left( \frac{\delta T_{abs,i,j}}{\delta b_C} \right)^2 \sigma_{b_C}^2 \quad (M.2) \]

\[ \frac{\delta T_{abs,i,j}}{\delta m_C} = \left( -m_T m_{R,i,j} DC_{i,j} - m_T b_{R,i,j} - b_T + b_C \right) \frac{1}{m_C^2} \quad (M.3) \]

\[ \frac{\delta T_{abs,i,j}}{\delta m_T} = \frac{m_{R,i,j} DC_{i,j}}{m_C} + \frac{b_{R,i,j}}{m_C} \quad (M.4) \]

\[ \frac{\delta T_{abs,i,j}}{\delta m_{R,i,j}} = \frac{m_T DC_{i,j}}{m_C} \quad (M.5) \]

\[ \frac{\delta T_{abs,i,j}}{\delta DC_{i,j}} = \frac{m_T m_{R,i,j}}{m_C} \quad (M.6) \]

\[ \frac{\delta T_{abs,i,j}}{\delta b_{R,i,j}} = \frac{m_T}{m_C} \quad (M.7) \]

\[ \frac{\delta T_{abs,i,j}}{\delta b_T} = \frac{1}{m_C} \quad (M.8) \]

\[ \frac{\delta T_{abs,i,j}}{\delta b_C} = -\frac{1}{m_C} \quad (M.9) \]
Appendix N

Skin Temperature Model

Discussed below is a proposed model for deriving the skin temperature of a body of water. The goal of this work is to develop and test a physics-based approach to modeling the interaction between an atmospheric mixture and a body of water. This approach is directly applicable to sea surface temperature modeling and is intended to provide a more accurate method of skin temperature extraction from bulk temperature and meteorological measurements. Previous models generally focus on two types of methods for predicting skin temperatures: skin thickness derivation and surface renewal. Both methods have shown to be accurate under limiting weather conditions. However, in the presence of high winds, extreme temperatures, and cloud cover these models break down and deviate from one another. In contrast to previous methods, the approach of this work is to examine the system from a first principles point of view and solve the energy, mass, and momentum balances across the system by obeying the conservation of mass, energy, and momentum. The proposed model seeks to perform on par or better than the previous models for ideal conditions as well as modeling skin temperature in non-deal conditions within a defined degree of uncertainty.

Previous models have typically been used in ocean temperature extraction applications. Errors reported by the reviewed models range from 0.1 – 0.5°C for oceanic applications. It is important to realize that from a big picture point of view this error is most likely acceptable. A final predicted value is only ever as accurate as the least accurate component in the system. For ocean applications both atmospheric modeling and satellite observations will probably contribute a larger uncertainty to the final predictions relative to the modeled skin temperatures uncertainty. However, with the advent of more and more airborne IR
platforms, the proposed model’s accuracy yields a benefit. Higher resolution systems, operated at low altitudes, provide the capability to detect high frequency changes in surface temperature as well as implications of localized weather phenomena. In addition, capturing imagery at low altitudes reduces the amount of atmospheric modeling required as well as the uncertainty it introduces. The improved accuracy and ability to predict implications of meteorological fluctuations on skin temperature make the proposed model more suited and advantageous for localized water modeling applications.

N.1 Energy flux through control volumes

In order to simplify the initial analysis, a very basic system is constructed. The system is defined as three control volumes\(^1\) stacked on top of one another. The construction of the system is depicted in figure N.1.

\(\text{Figure N.1: System definition: Control volume boundaries are defined by dotted lines.}\)

For the sake of simplified analysis, the sides of the control volumes in all directions, except for vertical, are considered adiabatic. It is assumed that no chemical reactions are

---

\(^1\) A control volume is defined as a fixed volume in space through which mass and fluid can flow.
taking place in any of the control volumes and kinetic and potential energy variations are ignored.

The first control volume \((CV_{gas})\) is filled with a mixture of dry air of an assumed composition and water vapor. The dry air/water vapor mixture, referred to as the gas in the future, is at pressure \(P_{gas}\) and temperature \(T_{gas}\). It is assumed the gaseous mixture behaves as an ideal, non-compressible gas and is optically thin \((\tau = 1.0)\). The mixture is analyzed using molar balances to allow for a simplified evaluation using gas equations-of-state, phase equilibrium constraints, and the ideal gas law. The total number of moles of water vapor and moles of dry air are defined as \(n_{vap}\) and \(n_{air}\), respectively. Therefore, the total number of moles, \(n_{gas}\), initially present within \(CV_{gas}\) is the summation of two constituents, shown in equation N.1.

\[
n_{gas} = n_{vap} + n_{air} \quad (N.1)
\]

The second control volume \((CV_{skin})\) encompasses the interfacial region between the gaseous mixture and pure water and is infinitesimally thin. The defined region is comprised of only its top surface in contact with the gaseous mixture and its bottom surface in contact with the liquid. Due to this definition, the number of moles of substance is inconsequential.

The third control volume \((CV_{bulk})\) contains only pure water, referred to as the liquid in the future, and is at pressure \(P_{liq}\) and temperature \(T_{skin}\). The total number of moles of liquid, \(n_{liq}\), is constant and infinite.

Any energy entering into a control volume is assumed positive while any energy leaving a control volume is assumed negative. Work done on a control volume is assumed positive and any work done by a control volume is assumed negative.

To predict the temperature of the skin control volume, independent differential equations will be produced from constructing the equations of state for all three volumes. These equations of state will be functionally dependent on measurable and derivable conditions. Each equation is solved simultaneously while stepping through time to produce a predicted value for skin temperature. Appropriate seed values for the unknown values in each equation must be chosen based on a process TBD.
N.2 Control Volume 1: Dry air-water vapor mixture

Due to the inability to measure the molar ratios of each gas constituent in the final application directly, a method is developed to calculate these values from measurable quantities. In order to determine the percentage of the two constituents present in the mixture, molar fractions are determined using the measured relative humidity and an application of Raoult’s Law.

Relative humidity, $\omega$, describes the amount of water vapor present in a gaseous mixture of air and water and is defined as the ratio of the partial pressure of water vapor in the mixture, $P_{vap}$, to the saturated vapor pressure of water at the given temperature, $P_{sat}$. Normally this value is given as a percentage and is expressed below in equation N.2.

$$\omega = \left( \frac{P_{vap}}{P_{sat}} \right) \times 100\% \quad \text{(N.2)}$$

Raoult’s Law is the equilibrium equation used for ideal gases and states that the vapor pressure of an ideal solution is dependent on the vapor pressure of each chemical component and the mole fraction of the component present in the solution. The mole fraction of a constituent is a value of expressing the composition of a mixture and is defined as the ratio of the amount of a constituent to the total amount of substance in the system. Applied to the mixture in $CV_{gas}$, the molar fractions for both the air and water vapor are defined in equations N.3 and N.4, respectively.

$$y_{air} = \frac{n_{air}}{n_{air} + n_{vap}} = \frac{n_{air}}{n_{gas}} \quad \text{(N.3)}$$

$$y_{vap} = \frac{n_{vap}}{n_{air} + n_{vap}} = \frac{n_{vap}}{n_{gas}} \quad \text{(N.4)}$$

For the given mixture, Raoult’s Law dictates that product of the partial pressure of a constituent will be equal to the mixture pressure, $P_{gas}$, and the molar fraction of each constituent. The assumption is made that the constituents are pure and have mole fractions equal to 1 in the liquid phase. Therefore the product of the mixture pressure and a constituent mole fraction will be equal to the vapor pressure of each constituent at the given temperature, $T_{gas}$, as shown in equations N.5 and N.6.
Combining equations N.3 and N.5 produces the relationship between $n_{vap}$, its related pressures, and the $n_{air}$ shown in equation N.7.

\[
\begin{align*}
    P_{air} &= P_{gas}y_{air} = P_{air}^{*}\bigg|_{T_{gas}} \Rightarrow y_{air} = \frac{P_{air}^{*}\bigg|_{T_{gas}}}{P_{gas}} \\
    P_{vap} &= P_{gas}y_{vap} = P_{vap}^{*}\bigg|_{T_{gas}} \Rightarrow y_{vap} = \frac{P_{vap}^{*}\bigg|_{T_{gas}}}{P_{gas}} 
\end{align*}
\] (N.5) (N.6)

\[
\begin{align*}
    n_{vap} &= \frac{P_{vap}^{*}}{P_{gas}} (n_{vap} + n_{air}) \\
    \Rightarrow n_{vap} \left(1 - \frac{P_{vap}^{*}}{P_{gas}}\right) &= \frac{P_{vap}^{*}}{P_{gas}} n_{air} \\
    \Rightarrow n_{vap} &= \frac{\frac{P_{vap}^{*}}{P_{gas}} n_{air}}{1 - \frac{P_{vap}^{*}}{P_{gas}}} 
\end{align*}
\] (N.7)

Through the ideal gas law, the derived expression for the number of moles of vapor, $n_{vap}$ is used to generate a relation with the number of moles of air, $n_{air}$. The ideal gas law, shown in equation N.8, describes the relationship of a gas’s state to its pressure, volume, and temperature.

\[
P_{gas}V_{gas} = n_{gas}RT
\] (N.8)

Equation N.8 can be rearranged to calculate the number of moles of air present in a given volume of gas. Using this relationship, and substituting equation N.7 for $n_{vap}$, an equation for $n_{air}$ (equation N.9) is produced. The derivation of this relation is shown below.
\[
P_{gas}V_{gas} = (n_{air} + n_{vap})RT_{\infty}
\]

\[
\Rightarrow n_{air} + n_{vap} = \frac{P_{gas}V_{gas}}{RT_{\infty}}
\]

\[
\Rightarrow \frac{P_{vap}^*}{P_{gas}}n_{air} + n_{air} = \frac{P_{gas}V_{gas}}{RT_{\infty}}
\]

\[
\Rightarrow n_{air} \left[ 1 + \frac{P_{vap}^*}{P_{gas}} \right] = \frac{P_{gas}V_{gas}}{RT_{\infty}}
\]

\[
\Rightarrow n_{air} = \frac{P_{gas}V_{gas}}{RT_{\infty} \left[ 1 + \frac{P_{vap}^*}{P_{gas}} \right]}
\]

The value for \( P_{vap}^* \) is equal to \( P_{vap} \) from the measured relative humidity (equation N.2) through Raoult’s Law. The size of \( CV_{gas} \) can be dictated by the application and environment which is being modeled. Therefore the number of moles of both constituents become functions of the defined volume, \( V_{gas} \).

\( CV_{gas} \) shares boundaries with \( CV_{skin} \) and the air column above. All other boundaries are currently ignored due to assumptions. If it is desired to include the air column in the overall system analysis, the optical transmission and physical properties of said volume need to be modeled externally (i.e. MODTRAN) and then married to \( CV_{skin} \). For consistency, the air column possible above \( CV_{gas} \) will be referred to as the atmospheric control volume (\( CV_{atm} \)).

It is assumed that the amount of air molecules present in the defined volume remains constant. Therefore, the rate of change of the air mass is zero, as shown below in equation N.10

\[
\frac{dn_{air}}{dt} = 0 \quad (N.10)
\]

Due to the shared boundary with the water volume interface, a change in the amount of water vapor within \( CV_{gas} \) can be induced through either evaporation or condensation at this interface. If the air column above \( CV_{gas} \) is incorporated then there can also be mass transport across the upper shared boundary. The rate of change in the number of
The rate of change of internal energy of $CV_{gas}$ is dictated by several phenomenon. Energy entering the control volume includes irradiance from $CV_{atm}$, $Q_{I:in}$, energy associated with mass evaporating from $CV_{skin}$, $Q_{evap}$, energy introduced by convection of the mixture over the water surface, $Q_{conv}$, and emissive energy from water surface, $Q_{E:in}$. Energy leaving $CV_{gas}$ includes irradiance into $CV_{skin}$ from $CV_{gas}$, $Q_{I:out}$, water vapor escaping to $CV_{atm}$, $Q_{vap}$, and emissive energy from $CV_{gas}$ to $CV_{atm}$, $Q_{E:out}$. Figure N.2 depicts all the incoming energy sources and outgoing energy sinks for $CV_{gas}$.

The overall governing energy balance, shown in equation N.13, can be reduced to a simplified form by making several assumptions.

$$\frac{d}{dt} (n_{vap} \bar{U}_{vap} + n_{air} \bar{U}_{air}) = (Q_{I:in} - Q_{I:out}) + (Q_{E:in} - Q_{E:out}) + Q_{conv_{skin->gas}} + Q_{evap_{skin->gas}} - Q_{vap_{gas->atm}}$$ (N.13)

Based on the system definition, it was assumed that $CV_{gas}$ was an optically thin volume of gas having perfect transmission ($\tau = 1.0$). Therefore $Q_{I:in} = Q_{I:out}$ and both terms drop from equation N.13. By the same reasoning both of the emissive terms will be equal, $Q_{E:in} = Q_{E:out}$, and can be dropped from the equation. Applying the above assumptions and expanding the remaining terms produces the following equation, equation N.14.
Due to another initial assumption that kinetic and potential energy variations will be ignored, both of these terms drop from both mass transfer rate equations. In addition the left side of the equation can be expanded and simplified. For a gas assumed ideal, specific internal energy depends on only temperature, therefore the specific heat $c_v$ is also a function of only temperature, as shown in equation N.15.

$$c_v = \left( \frac{\delta u}{\delta T} \right)_v \rightarrow c_v = \frac{du}{dT} \quad \text{(N.15)}$$

Applying the above mentioned simplifications produces the following expression for
the energy balance in $CV_{\text{gas}}$.

\[
\frac{dn_{\text{vap}}}{dt} \frac{d\bar{U}_{\text{vap}}}{dt} + \frac{dn_{\text{air}}}{dt} \frac{d\bar{U}_{\text{air}}}{dt} = \dot{h}(T_{\text{skin}} - T_{\text{gas}}) + \dot{n}_{\text{vap skin}}(H_{\text{skin}}) - \dot{n}_{\text{vap gas atm}}(H_{\text{gas}})
\]

\[
\frac{dn_{\text{vap}}}{dt} C_{\text{vap}} \frac{dT_{\text{gas}}}{dt} + \frac{dn_{\text{air}}}{dt} C_{\text{air}} \frac{dT_{\text{gas}}}{dt} = \dot{h}(T_{\text{skin}} - T_{\text{gas}}) + \dot{n}_{\text{vap skin}}(H_{\text{skin}}) - \dot{n}_{\text{vap gas atm}}(H_{\text{gas}})
\]

\[
\frac{dT_{\text{gas}}}{dt} (\dot{n}_{\text{vap}} C_{\text{vap}} + \dot{n}_{\text{air}} C_{\text{air}}) = \dot{h}(T_{\text{skin}} - T_{\text{gas}}) + \dot{n}_{\text{vap skin}}(H_{\text{skin}}) - \dot{n}_{\text{vap gas atm}}(H_{\text{gas}})
\]

(N.16)

N.3 Control Volume 2: Gas/water interface

It is assumed that the mass remains constant within $CV_{\text{skin}}$ and therefore any mass entering the volume must exit the volume by conservation of mass. Because the water is the only substance capable in this system to cross into $CV_{\text{skin}}$, the mass balance is simple to analyze. Mass entering the interfacial region from the water volume below must exit $CV_{\text{skin}}$ as vapor through the process of evaporation. The evaporation term leaving $CV_{\text{skin}}$ is identical to the evaporation term entering $CV_{\text{gas}}$ in the energy and mass balances. It is important to note that the mass transfer across the $CV_{\text{gas}}/CV_{\text{skin}}$ boundary is negative. This negation indicates the direction of movement is from $CV_{\text{skin}}$ into $CV_{\text{gas}}$. If the environment were to dictate vapor from the $CV_{\text{gas}}$ is condensing onto $CV_{\text{skin}}$ the phenomenon change would be induced by reversing the signs in equation N.21. By extension, it can be concluded that a reversal of the evaporation process would indicate the renewal of water molecules from the bulk volume to the skin would be reversed under these conditions.
The rate of change of internal energy of $CV_{skin}$ is dictated by several phenomenon. Energy entering the control volume includes irradiance which passed through $CV_{atm}$, $Q_{I,in}$, conductive energy from the control volume of water below, $Q_{cond}$, and energy associated with mass brought up from $CV_{bulk}$ to renew evaporated vapor molecules, $Q_{renew}$. Energy leaving $CV_{skin}$ includes energy emitted into $CV_{gas}$, $Q_{E,out}$, energy convectively removed by the gas above, $Q_{conv}$, and energy associated from water molecules evaporating at the top boundary of $CV_{skin}$ into $CV_{gas}$, $Q_{evap}$. The figure below, figure N.3, depicts all the incoming energy sources and outgoing energy sinks.

\[
\frac{dn_{liq}}{dt} = -\dot{n}_{skin\rightarrow gas} - \dot{n}_{bulk\rightarrow skin} \\
0 = -\dot{n}_{skin\rightarrow gas} - \dot{n}_{bulk\rightarrow skin} \\
-\dot{n}_{skin\rightarrow gas} = \dot{n}_{bulk\rightarrow skin} \quad \text{(N.17)}
\]

Figure N.3: Incoming and outgoing energy sources for control volume skin

Similar to the energy balance for $CV_{gas}$, the energy balance for $CV_{skin}$ can be simplified. The overall energy balance is shown below in equation N.18.
\[
\frac{d}{dt}(m_{\text{liq}} U_{\text{liq}}) = Q_{I: \text{in}} - Q_{E: \text{in}} - Q_{\text{conv, skin} \rightarrow \text{gas}}
\]
\[
- Q_{\text{evap, skin} \rightarrow \text{gas}} + Q_{\text{renew, bulk} \rightarrow \text{skin}} + Q_{\text{cond, bulk} \rightarrow \text{skin}} \quad \text{(N.18)}
\]

Expansion of the terms in the overall energy balance yields the following results. To reduce redundancy, the kinetic and potential energies have been removed from the evaporation renewal terms.

\[
\dot{m}_{\text{liq}} \frac{dU_{\text{liq}}}{dt} = Q_{I: \text{in}} - Q_{E: \text{in}}
\]
\[
- \dot{h}_{\text{gas}} (T_{\text{skin}} - T_{\text{gas}})
\]
\[
- \dot{m}_{\text{skin} \rightarrow \text{gas}} (h_{\text{skin}})
\]
\[
+ \dot{m}_{\text{bulk} \rightarrow \text{skin}} (h_{\text{bulk}})
\]
\[
+ \frac{k_{\text{bulk}}}{d \tau_{\text{bulk}}} (T_{\text{skin}} - T_{\text{bulk}}) \quad \text{(N.19)}
\]

It is important to note that if condensation was occurring the term for evaporation, \(-\dot{m}_{\text{skin} \rightarrow \text{gas}} (h_{\text{skin}})\), would be replaced with \(+\dot{m}_{\text{gas} \rightarrow \text{skin}} (h_{\text{vap}})\). In addition, similar to the ideal gas assumption, when a liquid is idealized it is assumed to be incompressible and its specific internal energy depends only on temperature. Therefore, the specific heat of the liquid will also only depend on temperature, as depicted in equation N.15. This assumption simplifies equation N.19 to equation N.20.

\[
\dot{m}_{\text{liq}} C_v (T_{\text{bulk}}) = Q_{I: \text{in}} - Q_{E: \text{in}}
\]
\[
- \dot{h}_{\text{gas}} (T_{\text{skin}} - T_{\text{gas}})
\]
\[
- \dot{m}_{\text{skin} \rightarrow \text{gas}} (h_{\text{skin}})
\]
\[
+ \dot{m}_{\text{bulk} \rightarrow \text{skin}} (h_{\text{bulk}})
\]
\[
+ \frac{k_{\text{bulk}}}{d \tau_{\text{bulk}}} (T_{\text{skin}} - T_{\text{bulk}}) \quad \text{(N.20)}
\]
N.4 Control Volume 3: Pure water

The final control volume, $CV_{bulk}$, is comprised of only water and is assumed to maintain a constant mass. Molecules leaving the control volume via evaporation are replaced by water molecules brought up from the water located below the bulk volume. By conservation of mass, both the rate of evaporation from the volume and the rate of renewal of molecules into the volume must be equal.

$$\frac{dn_{liq}}{dt} = 0 = \dot{n}_{renew} - \dot{n}_{evap}$$

$$\dot{n}_{\infty\rightarrow bulk} = \dot{n}_{bulk\rightarrow skin}$$  (N.21)

The rate of change of internal energy within $CV_{liq}$ is dictated by the removal of energy via conduction with the skin volume, $Q_{cond}$, and any energy brought in through the renewal process and then removed via evaporation, $Q_{renew}$. The figure below, figure N.4, depicts all the incoming energy sources and outgoing energy sinks.

![Figure N.4: Incoming and outgoing energy sources for control volume liquid](image)

Figure N.4: Incoming and outgoing energy sources for control volume liquid
The overall energy balance for $CV_{liq}$ is shown below in equation N.22.

$$\frac{d}{dt} (U_{liq}) = Q_{\text{renew}} - Q_{\text{evap}} - Q_{\text{cond}}$$

$$= \dot{n}_{\infty \rightarrow \text{bulk}} h_{\infty} - \dot{n}_{\text{bulk} \rightarrow \text{skin}} h_{\text{bulk}} - \frac{k_{liq}}{dT_{\text{bulk}}} (T_{\text{skin}} - T_{\text{bulk}})$$

$$= \dot{n}_{\text{bulk} \rightarrow \text{skin}} (h_{\infty} - h_{\text{bulk}}) - \frac{k_{liq}}{dT_{\text{bulk}}} (T_{\text{skin}} - T_{\text{bulk}})$$

$$= \dot{n}_{\text{bulk} \rightarrow \text{skin}} [c_v (T_{\infty} - T_{\text{bulk}})] - \frac{k_{liq}}{dT_{\text{bulk}}} (T_{\text{skin}} - T_{\text{bulk}}) \quad (N.22)$$
Appendix O

Swarm Repeatability Results

The data presented below is the full data set to support Table 6.5.

![Graph](image)

Figure O.1: Comparison of initial and final ice fraction correlations for the Trial 1.
Figure O.2: Comparison of initial and final ice fraction correlations for the Trial 2.

Figure O.3: Comparison of initial and final ice fraction correlations for the Trial 3.
Figure O.4: Comparison of initial and final ice fraction correlations for the Trial 4.

Figure O.5: Comparison of initial and final ice fraction correlations for the Trial 5.
Figure O.6: Comparison of initial and final ice fraction correlations for the Trial 6.

Figure O.7: Comparison of initial and final ice fraction correlations for the Trial 7.
Figure O.8: Comparison of initial and final ice fraction correlations for the Trial 8.

Figure O.9: Comparison of initial and final ice fraction correlations for the Trial 9.
CHAPTER O. Swarm Repeatability Results

Figure O.10: Comparison of initial and final ice fraction correlations for the Trial 10.

Figure O.11: Comparison of initial and final ice fraction correlations for the Trial 11.
Figure O.12: Comparison of initial and final ice fraction correlations for the Trial 12.

(a) Initial ice fraction correlation

(b) Final ice coverage correlation

Figure O.13: Swarm flow rates for Trial 1.

(a) Initial ice fraction correlation

(b) Final ice coverage correlation
CHAPTER O. Swarm Repeatability Results

(a) Initial ice fraction correlation

(b) Final ice coverage correlation

Figure O.14: Swarm flow rates for Trial 2.

(a) Initial ice fraction correlation

(b) Final ice coverage correlation

Figure O.15: Swarm flow rates for Trial 3.
Figure O.16: Swarm flow rates for Trial 4.

Figure O.17: Swarm flow rates for Trial 5.
Figure O.18: Swarm flow rates for Trial 6.

Figure O.19: Swarm flow rates for Trial 7.
Figure O.20: Swarm flow rates for Trial 8.

(a) Initial ice fraction correlation

(b) Final ice coverage correlation

Figure O.21: Swarm flow rates for Trial 9.

(a) Initial ice fraction correlation

(b) Final ice coverage correlation
CHAPTER O. Swarm Repeatability Results

(a) Initial ice fraction correlation

(b) Final ice coverage correlation

Figure O.22: Swarm flow rates for Trial 10.

(a) Initial ice fraction correlation

(b) Final ice coverage correlation

Figure O.23: Swarm flow rates for Trial 11.
(a) Initial ice fraction correlation  

(b) Final ice coverage correlation

Figure O.24: Swarm flow rates for Trial 12.
Appendix P

ALGE input files

Below are samples of all the ALGE input files used for this application. It should be noted that some of the files are not included in their entirety due to their length. This section is intended to offer examples of the files formats and the data that is expected in the various inputs.

P.1  param.dat

42.5
DX (GRID SPACING IN X-DIR IN M)
42.5
DY (GRID SPACING IN Y-DIR IN M)
0.5
dz (vertical resolution, m)
0.0
TIME (START TIME, LOCAL, HOURS)
2820.01
TMAX (TOTAL RUN TIME, HOURS)
43.6
RLAT (LATITUDE, DEGREES)
-84.25
RLON (LONGITUDE, DEGREES)
336.0
DAY (JULIAN DAY)
0.001
Z0 (ROUGHNESS, METERS)
0.001
z0m (marsh roughness length, m)
0.5
TBIN (BOUNDARY INFLOW TEMP DEG C)
0.5
TBOU (BOUNDARY OUTFLOW TEMP DEG C)
3.0
TSRS (MASS SOURCE TEMP, DEG C)
20.7
SRSFL (MASS SOURCE, M**3/S)
20.7
SNKFL (MASS SINK, M**3/S)
180.0
XAX (ANGLE OF +X AXIS FROM NORTH, DEG)
3600.0
PINT (PRINT INTERVAL IN SECONDS)
10.
ZREF (REFERENCE HEIGHT FOR TURBULENT SURFACE LAYER, METERS)
0.0
dzdx
0.0
dzdy
0.2
tsmlt (multiplier for Courant limit)
0.1
dtinit (initial timestep in seconds)
0.0
uinit
0.0
vinit
0.0
sinit (initial salinity over entire domain)
5
ihtn (# timesteps between calls to heat transfer subroutines)
39
NX (# OF NODES IN X-DIR)
71
NY (# OF NODES IN Y-DIR)
0
INDG (SWITCH FOR USE OF NUDGING DATA, 0=OFF, 1=ON)
1
lsrs (grid level of mass source counting down from surface)
1
ihttr (flag for heat transfer functions, 0=off, 1=on)
1
ltmpsrg (flag for outfall temp. 0 = fixed, 1 = delta-t, deg C)
10
irhm (max. # 2-D surface wave loops per 3-D loop)
1
idatsrs2 (# hours of second mass source data (m**3/s))
irestart (flag for restart option, 0 = no, 1 = yes)
i = 1
ikopt (flag for Yamada TKE mixing (0), or molecular viscosity (1))
i = 0
isal (flag for fresh (0) or salt (1) water)
i = 1
iflow (flag for time-dependent (1) or steady-state primary mass source (0))
P.3  igrid.dat

00000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000011166611100000000000000000000000
00000000000000000000000000000000000000011111111111000000000000000000000
00000000000000000000000000000000011111101111111111110000000000000000000
00000000000000000000000000000000011111110111111111111100000000000000000
00000000000000000000000000000000111111111011111111111110000000000000000
00000000000000000000000000000000000000011111111111111111111100000000000
00000000000000000000000000000000000000000000111111111111111111111100000
00000000000000000000000000000000000000000000001111111111111111111111100
00000000000000000000000000000000000000000000000000000000000000000000000
P.4 flow.dat

Each row of data represents an hour of simulation time. The flow file requires an entry for each hour being simulated. The example below is only a small sample and is meant as an example only.

20.7
20.7
20.7
20.7
20.7
20.7
20.7
20.7
20.7
20.7
20.7
20.7
20.7
20.7
20.697431
20.699955
20.468378
20.705003
21.038802
21.088651
20.869694
20.937211
21.084865
21.105057
21.025551
20.924591
20.725826
20.57691
20.622342
20.566183
20.470271
20.565552
20.617294
20.752959
20.705634
20.636855
20.643796
P.5  deltat.dat

Each row of data represents an hour of simulation time. The deltaT file requires an entry for each hour being simulated. The example below is only a small sample and is meant as an example only.

4.07268
4.07268
4.07268
4.07268
4.07268
4.07268
4.07268
4.07268
4.07268
4.07268
3.91797
3.91797
3.91797
3.91797
3.91797
3.91797
3.91797
3.91797
3.91797
3.91797
P.6  sfc.dat

Each row of data represents an hour of simulation time. The sfc file requires an entry for each hour being simulated. The example below is only a small sample and is meant as an example only.

2900
6 10 1.54 1 1 0.15 987.47 0.03 12/1/2008 5:00
7 10 2.57 1 1 0.15 986.8 0.03 12/1/2008 6:00
8 10 2.06 1 1 0.15 986.8 0.03 12/1/2008 7:00
9 0 0.5 1 1 0.75 0.15 986.8 0.03 12/1/2008 7:59
10 330 1.54 1 1 0.15 986.46 0.03 12/1/2008 9:00
11 330 2.57 1 0 0.375 0.21 986.46 0.03 12/1/2008 10:00
12 340 1.54 1 -1 0.75 0.39 987.14 0.03 12/1/2008 10:59
13 330 2.06 0 -1 0.75 0.33 988.15 0.03 12/1/2008 12:00
14 310 3.09 0 -1 0.375 0.39 987.14 0.03 12/1/2008 12:59
15 320 3.09 0 -1 0.375 0.36 989.17 0.03 12/1/2008 13:59
16 310 3.09 0 -2 1 1.5 990.86 0.03 12/1/2008 15:00
17 310 3.6 0 -2 0.375 0.48 991.2 0.03 12/1/2008 16:00
18 310 3.6 -1 -3 0.375 0.51 992.55 0.03 12/1/2008 16:59
19 300 4.12 -1 -2 0.375 1.17 993.57 0.03 12/1/2008 18:00
20 290 3.09 -1 -3 0.375 0.54 994.59 0.03 12/1/2008 19:00
21 260 3.09 -1 -2 1 0.36 996.28 0.03 12/1/2008 19:59
22 270 3.6 -1 -2 0.75 0.39 996.96 0.03 12/1/2008 21:00

P.7  dimar.inc

c input array dimar.inc
c fixed dimensions, set with parameter statements
c nxa = nodes in x-dir
c nya = nodes in y-dir
c nza = nodes in z-dir
c nmet = hours of meteorological data
c nsra = hours of time series output to be stored
c ndta = hours of delta-T data
c nba = # of nodes at boundary with inflow or outflow
parameter (nxa =39)
parameter (nya =71)
parameter (nza = 9)
parameter (nmet = 10000)
parameter (nsra = 10000)
parameter (ndta = 10000)
parameter (nba = 500)
parameter (nsr = 500)
parameter (nsn = 500)
| 1 | 0.00081 0.33402 0.50812 0.52535 0.52545 0.52555 0.52565 0.52575 0.52585 0.52595 0.52605 |
| 2 | 0.00081 0.32935 0.49983 0.52111 0.52121 0.52131 0.52141 0.52151 0.52161 0.52171 0.52181 |
| 3 | 0.00081 0.32468 0.49155 0.51686 0.51696 0.51706 0.51716 0.51726 0.51736 0.51746 0.51756 |
| 4 | 0.00080 0.32002 0.48327 0.51262 0.51272 0.51282 0.51292 0.51302 0.51312 0.51322 0.51332 |
| 5 | 0.00080 0.31535 0.47498 0.50838 0.50848 0.50858 0.50868 0.50878 0.50888 0.50898 0.50908 |
| 6 | 0.00080 0.31068 0.46670 0.50130 0.50140 0.50150 0.50160 0.50170 0.50180 0.50190 0.50200 |
| 7 | 0.00080 0.30602 0.45841 0.49989 0.50009 0.50019 0.50029 0.50039 0.50049 0.50059 0.50069 |
| 8 | 0.00079 0.30135 0.45013 0.49565 0.50005 0.50145 0.50155 0.50165 0.50175 0.50185 0.50195 |
| 9 | 0.00079 0.29668 0.44184 0.49140 0.50174 0.50184 0.50194 0.50204 0.50214 0.50224 0.50234 |
| 10 | 0.00079 0.29202 0.43356 0.48716 0.51971 0.51981 0.51991 0.52001 0.52011 0.52021 0.52031 |
| 11 | 0.00078 0.28735 0.42528 0.48292 0.52068 0.52078 0.52088 0.52098 0.52108 0.52118 0.52128 |
| 12 | 0.00078 0.28268 0.41699 0.47868 0.52164 0.52174 0.52184 0.52194 0.52204 0.52214 0.52224 |
| 13 | 0.00079 0.28756 0.42224 0.49055 0.54084 0.54717 0.54727 0.54737 0.54747 0.54757 0.54767 |
| 14 | 0.00079 0.29249 0.42704 0.50094 0.55749 0.56744 0.56754 0.56764 0.56774 0.56784 0.56794 |
| 15 | 0.00080 0.29742 0.43184 0.51133 0.57414 0.58770 0.58780 0.58790 0.58800 0.58810 0.58820 |
| 16 | 0.00080 0.30235 0.43665 0.52173 0.59080 0.60797 0.60807 0.60817 0.60827 0.60837 0.60847 |
| 17 | 0.00081 0.30728 0.44145 0.53212 0.60745 0.62824 0.62834 0.62844 0.62854 0.62864 0.62874 |
| 18 | 0.00082 0.31221 0.44625 0.54251 0.62410 0.64850 0.64860 0.64870 0.64880 0.64890 0.64900 |
| 19 | 0.00082 0.31714 0.45106 0.55291 0.64075 0.66877 0.66887 0.66897 0.66907 0.66917 0.66927 |
| 20 | 0.00083 0.32207 0.45586 0.56330 0.65740 0.68904 0.68914 0.68924 0.68934 0.68944 0.68954 |

<table>
<thead>
<tr>
<th>P.9 peramp.dat</th>
</tr>
</thead>
<tbody>
<tr>
<td>1664</td>
</tr>
<tr>
<td>6.0 0.0</td>
</tr>
<tr>
<td>6.0 0.0</td>
</tr>
<tr>
<td>6.0 0.0</td>
</tr>
<tr>
<td>6.0 0.0</td>
</tr>
<tr>
<td>6.0 0.0</td>
</tr>
<tr>
<td>6.0 0.0</td>
</tr>
<tr>
<td>6.0 0.0</td>
</tr>
<tr>
<td>6.0 0.0</td>
</tr>
<tr>
<td>6.0 0.0</td>
</tr>
<tr>
<td>6.0 0.0</td>
</tr>
<tr>
<td>6.0 0.0</td>
</tr>
<tr>
<td>6.0 0.0</td>
</tr>
<tr>
<td>6.0 0.0</td>
</tr>
<tr>
<td>6.0 0.0</td>
</tr>
<tr>
<td>6.0 0.0</td>
</tr>
<tr>
<td>6.0 0.0</td>
</tr>
<tr>
<td>6.0 0.0</td>
</tr>
<tr>
<td>6.0 0.0</td>
</tr>
<tr>
<td>6.0 0.0</td>
</tr>
<tr>
<td>6.0 0.0</td>
</tr>
<tr>
<td>6.0 0.0</td>
</tr>
<tr>
<td>6.0 0.0</td>
</tr>
<tr>
<td>6.0 0.0</td>
</tr>
</tbody>
</table>
P.10  seadens.dat

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1.00015</td>
<td>1.00056</td>
<td>1.00097</td>
<td>1.00138</td>
<td>1.00179</td>
<td>1.00220</td>
<td></td>
</tr>
<tr>
<td>1.00302</td>
<td>1.00629</td>
<td>1.00711</td>
<td>1.00751</td>
<td>1.00792</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.00874</td>
<td>1.00914</td>
<td>1.00955</td>
<td>1.00996</td>
<td>1.01037</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.00261</td>
<td>1.00302</td>
<td>1.00343</td>
<td>1.00384</td>
<td>1.00425</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.00548</td>
<td>1.00588</td>
<td>1.00629</td>
<td>1.00670</td>
<td>1.00711</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.00833</td>
<td>1.00874</td>
<td>1.00914</td>
<td>1.00955</td>
<td>1.00996</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

P.11  srsfl2.dat

For this application there was now second source so this file only contained a single value, 1.0.
Appendix Q

Computing cluster submission scripts

The work documented was performed on Rochester Institute of Technology’s Research Computing (RC) computing cluster. RC maintains a ‘condominium’ cluster consisting of shares purchased by researchers. Each node has 32-64 cores with AMD Opteron 2.2 GHz or AMD Interlagos 2.6 "bulldozer" processors and 128-256 GB of memory. There are currently over 250 cores in the cluster. Each node is interconnected with 10 Gigabit ethernet. The head node is connected with 10 Gigabit ethernet to the campus backbone. Aside from local scratch space, the filesystems are NSF mounted from the research computing fileserver, a BlueArc NAS system with 70 TB of usable space. This linux cluster is designed to run parallel computing jobs that are tightly coupled and use the MPI [1]. Through the course of this research the job management system changed from Sun Grid Engine (SGE) to the Simple Linux Utility for Resource Management (SLURM). Two versions of submission scripts to execute PSO-ALGE on the cluster were created to operate in the two environments.

Q.1 SLURM submission scripts

Q.1.1 config.sh

```bash
#!/bin/bash
```

195
if [ "$BASH_SOURCE" == $0 ] ; then echo "This is a config file, you don’t run it" ;
exit 1 ; fi

##Slurm config options##

#QoS to run in
SLURM_QOS="salvaggio-normal"

#Slurm partition
SLURM_PARTITION="work"

#Memory requirement PER JOB (total, not per job step)
SLURM_MEMORY_REQ="1024"

#Runtime limit PER JOB
SLURM_WALLCLOCK="2:00:0"

##Job Options##

#A prefix for all the log files, job names etc. Spaces are bad.
JOB_NAME='2W12'

#How many generations the job will run for (how deep the job is)
TOTAL_GENERATIONS=200

#How many workers will run per generation (how wide the job is)
NUM_STEPS=24

#How many processors each step will consume.
THREADS_PER_STEP=1

#This is the script that is called for ever step. It is passed two options
# The first option is a number (index 0) which is the generation
# The second option is a number (index 0) which is the step.
# Generation 5, worker 2’s run will look like "generation-step.sh 5 2"
STEP_WORKER_SCRIPT='/home/mva7609/may_casterline/SLURM/two_week_runs/0809/constant_flowrate/every_12hr/generation-step.sh'

JOB_FILE_DIRECTORY='./jobfiles'
LOG_FILE_DIRECTORY='./logs'

##Swarm Options##

# dataPath = where new data is going to reside
# alge_constant_path = path containing all ALGE input files and ground truth data
# name = the job name given to each particle in the scheduler
alge_constant_path='/home/mva7609/may_casterline/SLURM/two_week_runs/0809/alge_12hr/'
dataPath='/home/mva7609/may_casterline/SLURM/two_week_runs/0809/constant_flowrate/every_12hr/data/'
name=$JOB_NAME

# num_particles_per_gen = how many ALGE instances will run per generation
num_particles_per_gen=$NUM_STEPS
# num_parameters = how many ALGE inputs that are being optimized
# =number of points to average the flow file to
# --> if only optimizing flow (weather_variable=0)
# --> this value is equal to the total number of points
# in the flow file, divided by the window size used
# to create the new flow array
# EXAMPLE: Flow file has 2900 points and user wants
# a flow value to be calculated every 145 points.
# 145 is the window size, so there will be 20
# entries in the flow file, representing a value
# approximately every 6 days (assuming the time
# resolution is hourly).
# =8 --> if only optimizing weather (weather_variable=1)
# =10 --> optimizing both weather and flow (weather_variable=2)
# num_parameters=20
#
# num_generations = how many generations the swarm will run for
# num_generations=$TOTAL_GENERATIONS
#
# Swarm parameters
# error_goal --> Used to terminate the algorithm
# minflag
# =0 --> converge on score array to be within error goal of 0.0
# =1 --> converge on score array to be within error goal of minimum score acheived
# gamma1 --> cognitive acceleration, relates to particle’s personal best solution
# gamma2 --> social acceleration, relates to global best solution
# w_start --> value of velocity at beginning
# w_end --> value of velocity at end
# w_varyfor --> the fraction of maximum iterations for which the velocity is linearly decreased
# error_goal=0.0001
# minflag=1
# gamma1=2.05
# gamma2=2.05
# w_start=1.2
# w_end=0.0
# w_varyfor=0.7
#
# ub = upper bound
# lb = lower bound
# These bounds are the bounding condition ranges for different modes of operation.
# If optimizing weather or weather and plant parameters these
values represent the range of possible % changes made to the overall time series.

# If optimizing only flow then these bounds represent the range the flow rate is allowed to fluctuate within at any point in time.

# initial_fwhm = initial full width half max for the gaussian distribution of initial flow rates
# initial_mean = initial mean for the gaussian distribution of initial flow rates

ub=45.0
lb=1.0
initial_fwhm=4.0
initial_mean=5.0

# op_mode decides which evaluation module is run
# =0 --> weather parameters are considered valid and left alone, only flow is optimized
# =1 --> weather parameters are the only things optimized
# =2 --> weather and plant parameters are optimized using % change to the overall time series
# =3 --> a 2D, single global minimum, test function is optimized with 2 parameters, x and y
# =4 --> every time point in a flow rate series is considered a parameter to optimize, resulting in an extremely high dimensional solution
# =5 --> evaluates a 2 week alge simulation using synthesized data as the truth set. Only optimizes flow rate every 18 hours and only evaluates ice fraction performance

op_mode=5

# ratio_flag = determine metric for evaluation
# 1 = ice only
# 2 = water only
# 3 = combination metric

ratio_flag=1

# metric_flag = determine metric used
# 1 = Modified RMS
# 2 = Standard RMS

metric_flag=2

# season = define which season of data to simulate
# 0 = 08/09 winter
# 1 = 09/10 winter

season=0
Q.1.2 generation-step.sh

```bash
#!/bin/bash

#Run the job from the current working directory
#$ -cwd

#SBATCH -p work

#Your commands go after this line

#Source files to pull variables from
source ~/.bashrc
source config.sh
ulimit -a
ulimit -n 4096 -u 4096

#Load IDL/ENVI binaries
module load envi

#Store first two incoming arguments as generation and particle
generation=$1
particle=$2

config_file='readlink -f config.sh'
config_file=""'
config_file=""'
dataPath=""'

#Push to directory containing IDL code
pushd /home/mva7609/may_casterton/PSO_Cluster_SLURM/

#Start IDL and pass in a set of commands
idl <<EOF
.compile pso_cluster_final_truth
resolve_all
value = pso_cluster(${generation}, ${particle}, ${config_file}, ${dataPath})
print, value
EOF

#Pop out of directory and exit
popd

echo -------
exit
```
#!/bin/bash

#TODO
#release generation 0 upon completion
#have a trigger on success to cancel remaining jobs

#Pull the config then do sanity tests
if [ -e config.sh ]; then source config.sh; else echo config file missing!; exit 1; fi

if [ -z "$JOB_NAME" ]; then echo "JOB_NAME not defined, this is used to prefix all jobs and logs (it should be unique per data set)"; exit 1; fi

if [ -z "$JOB_FILE_DIRECTORY" ]; then echo JOB_FILE_DIRECTORY not defined, this is where the job files are kept; exit 1; fi

if [ ! -d ${JOB_FILE_DIRECTORY} ]; then echo creating job file directory: ${JOB_FILE_DIRECTORY}; mkdir -p ${JOB_FILE_DIRECTORY}; fi

if [ -z "$LOG_FILE_DIRECTORY" ]; then echo Warning: $LOG_FILE_DIRECTORY not defined, this is where the log files are kept.; echo You have 15 seconds.; sleep 15; fi

if [ ! -d ${LOG_FILE_DIRECTORY} ]; then echo LOG_FILE_DIRECTORY $LOG_FILE_DIRECTORY not found, creating; mkdir -p ${LOG_FILE_DIRECTORY}; fi

if [ -z "$TOTAL_GENERATIONS" ]; then echo TOTAL_GENERATIONS not defined, this is how many generations deep the job is."; exit 1; fi

if [ -z "$NUM_STEPS" ]; then echo NUM_STEPS not defined, this is how many workers run per generation."; exit 1; fi

if [ -z "$THREADS_PER_STEP" ]; then echo "THREADS_PER_STEP not defined, this is how many processors each step requires."; exit 1; fi

if [ -z "$STEP_WORKER_SCRIPT" ]; then echo STEP_WORKER_SCRIPT not defined, this is what is run on every job step."; exit 1; fi

#End sanity tests
#############################################################################
#Below here are the loops to generate the generational job files with
#The steps inside them
#if any job files exit with this prefix, fail
if [ "$ls jobfiles/$(JOB_NAME)_*.sh 2> /dev/null" ]; then echo "Error, job files exist with this prefix"; exit 1; fi

#Begin Generation Loop#
for GENERATION in $(seq 0 $(expr $(TOTAL_GENERATIONS) - 1)); do


Q.1. SLURM submission scripts

```bash
#!/bin/bash -l

# NOTE the -l flag!

# This is an example job file for a single core CPU bound program
# Note that all of the following statements below that begin
# with #SBATCH are actually commands to the SLURM scheduler.
# Please copy this file to your home directory and modify it
# to suit your needs.

# If you need any help, please email rc-help@rit.edu

# Name of the job
#SBATCH -J ${JOB_NAME}_${GENERATION}

# Standard out and Standard Error output files
#SBATCH -o ${LOG_FILE_DIRECTORY}/${JOB_NAME}_${GENERATION}.stdout
#SBATCH -e ${LOG_FILE_DIRECTORY}/${JOB_NAME}_${GENERATION}.stderr

# Runtime Required
#SBATCH -t ${SLURM_WALLCLOCK}

#QOS to run under
#SBATCH --qos=${SLURM_QOS}

#Partition and CPUs required
#SBATCH -p ${SLURM_PARTITION} -n ${NUM_STEPS} -c ${THREADS_PER_STEP}

# Job memory requirements in MB
#SBATCH --mem=${SLURM_MEMORY_REQ}

# A check to see if this script is managed by SLURM
/usr/bin/env | grep SLURM_JOB_ID

if [ "$?" != "0" ]; then
    echo "Please run this script with 'sbatch <script-name>'"
    echo "Email rc-help@rit.edu if you have any questions."
    echo "Aborting."
else
    #Begin Step Loop#
    for STEPNUMBER in $(seq 0 $(expr ${NUM_STEPS} - 1) ); do
        #EOF
```

Q. Computing cluster submission scripts

Q.2 SGE submission scripts

Q.2.1 config.sh

```bash
#!/bin/bash

if [ "$BASH_SOURCE" == $0 ]; then echo "This is a config file, you don’t run it" ;
    exit 1 ; fi

##Slurm config options##

#QoS to run in
SLURM_QOS="rc-normal"

#Slurm partition
SLURM_PARTITION="work"

#Memory requirement PER JOB (total, not per job step)
SLURM_MEMORY_REQ="30"

#Runtime limit PER JOB
SLURM_WALLCLOCK="0:10:0"

##Job Options##
```
Q.2. SGE SUBMISSION SCRIPTS

17 # A prefix for all the log files, job names etc. Spaces are bad.
18 JOB_PREFIX='2WK'
19 # How many generations the job will run for (how deep the job is)
20 NUM_GENERATIONS=5
21 # How many workers will run per generation (how wide the job is)
22 NUM_STEPS=3
23 # How many processors each step will consume.
24 THREADS_PER_STEP=1
25
26 # This is the script that is called for ever step. It is passed two options
27 # The first option is a number (index 0) which is the generation
28 # The second option is a number (index 0) which is the step.
29 # Generation 5, worker 2's run will look like "generation-step.sh 5 2"
30 STEP_WORKER_SCRIPT="generation-step.sh"
31
32 ## Swarm Options ##
33
34 # dataPath = where new data is going to reside
35 # alge_constant_path = path containing all ALGE input files and ground truth data
36 # name = the job name given to each particle in the scheduler
37 alge_constant_path='/home/mva7609/may_casterline/SLURM/two_week_runs/alge/'
38 dataPath='./data/'
39 name=$JOB_PREFIX
40
41 # num_particles_per_gen = how many ALGE instances will run per generation
42 num_particles_per_gen=$NUM_STEPS
43
44 # num_parameters = how many ALGE inputs that are being optimized
45 # - number of points to average the flow file to
46 #   -- if only optimizing flow (weather_variable=0)
47 #   -- this value is equal to the total number of points
48 #   in the flow file, divided by the window size used
49 #   to create the new flow array
50 #   EXAMPLE: Flow file has 2900 points and user wants
51 #   a flow value to be calculated every 145 points.
52 #   145 is the window size, so there will be 20
53 #   entries in the flow file, representing a value
54 #   approximately every 6 days (assuming the time
55 #   resolution is hourly).
56 # -8 -- if only optimizing weather (weather_variable=1)
57 # -10 --> optimizing both weather and flow (weather_variable=2)
58 num_parameters=20
59
60 # num_generations = how many generations the swarm will run for
61 #num_generations=4
# Swarm parameters

## error_goal --> Used to terminate the algorithm

### minflag

- **0** --> converge on score array to be within error goal of 0.0
- **1** --> converge on score array to be within error goal of minimum score achieved

### gamma1 --> cognitive acceleration, relates to particle’s personal best solution

### gamma2 --> social acceleration, relates to global best solution

### w_start --> value of velocity at beginning

### w_end --> value of velocity at end

### w_varyfor --> the fraction of maximum iterations for which the velocity is linearly decreased

```plaintext
error_goal=0.0001
minflag=0
gamma1=2.05
gamma2=2.05
w_start=1.2
w_end=0.0
w_varyfor=0.7
```

## ub = upper bound

## lb = lower bound

- These bounds are the bounding condition ranges depending on the mode of operation.

### If optimizing weather or weather and plant parameters these values represent the range of possible % changes made to the overall time series.

### If optimizing only flow then these bounds represent the range the flow rate is allowed to fluctuate within at any point in time.

### initial_fwhm = initial full width half max for the gaussian distribution

### initial_mean = initial mean for the gaussian distribution of initial flow rates

### op_mode decides which evaluation module is run

- **0** --> weather parameters are considered valid and left alone, only flow is optimized
Q.2. SGE Submission Scripts

111 # =1 --> weather parameters are the only things optimized
112 # =2 --> weather and plant parameters are optimized using % change to
113 # the overall time series
114 # =3 --> a 2D, single global minimum, test function is optimized with 2
115 # parameters, x and y
116 # =4 --> every time point in a flow rate series is considered a parameter
117 # to optimize, resulting in an extremely high dimensional solution
118 # =5 --> evaluates a 2 week algae simulation using synthesized data as the
119 # truth set. Only optimizes flow rate every 18 hours and only evaluates
120 # ice fraction performance
121 op_mode=5
122
123 # ratio_flag = determine metric for evaluation
124 # 1 = ice only
125 # 2 = water only
126 # 3 = combination metric
127 # ratio_flag=1
128 # metric_flag = determine metric used
129 # 1 = Modified RMS
130 # 2 = Standard RMS
131 # metric_flag=2
132 # season = define which season of data to simulate
133 # 0 = 08/09 winter
134 # 1 = 09/10 winter
135 # season=0

Q.2.2 generation-step.sh

1 #!/bin/bash -l
2 #Run the job from the current working directory
3 #$ -cwd
4 #SBATCH -p work
5 #Your commands go after this line
6 #Source files to pull variables from
7 source ~/.bashrc
8 source config.sh
9 ulimit -a
10 ulimit -n 4096 -u 4096
11 #Load IDL/ENVI binaries
12 module load envi
#Store first two incoming arguments as generation and particle
generation=$1
particle=$2
cfg_file='readlink -f config.sh'
cfg_file="'"$cfg_file"'"
dataPath="'"${dataPath}"'"

dVal= 

#Push to directory containing IDL code
pushd /home/mva7609/may_castercle/PSO_Cluster_SLURM/

#Start IDL and pass in a set of commands
# 1. Compile main driving routine (pso_cluster_final_truth)
# 2. Compile any dependent routines
# 3. Execute main routine, passing into coming arguments as well as configuration
#    file variables
# 4. Print the returned value from the execution
# 5. End the IDL list of commands
IDL <<EOF
.compile pso_cluster_final_truth
.resolve_all
value = pso_cluster(${generation}, ${particle}, ${cfg_file}, ${dataPath})
.print, value
EOF

#Pop out of directory and exit
popd

echo -------
exit

Q.2.3 submit-generation.sh

#!/bin/bash

#TODO
#release generation 0 upon completion
#have a trigger on success to cancel remaining jobs

#Pull the config then do sanity tests
if [ -e config.sh ] ; then source config.sh ; else echo config file missing! ; exit 1 ; fi

if [ -z "$JOB_NAME" ] ; then echo "JOB_NAME not defined, this is used to prefix all
jobs and logs (it should be unique per data set)" ; exit 1 ; fi

if [ -z "$JOB_FILE_DIRECTORY" ] ; then echo JOB_FILE_DIRECTORY not defined, this is
where the job files are kept ; exit 1 ; fi

if [ ! -d ${JOB_FILE_DIRECTORY} ] ; then echo creating job file directory:
$JOB_FILE_DIRECTORY ; mkdir -p ${JOB_FILE_DIRECTORY} ; fi
if [ -z "$LOG_FILE_DIRECTORY" ]; then
echo "Warning: $LOG_FILE_DIRECTORY not defined, this is where the log files are kept.";
echo "You should ctrl-c a few times and fix this or else you may end up with lots of files in annoying places.";
echo "You have 15 seconds."; sleep 15; fi

if [ ! -d ${LOG_FILE_DIRECTORY} ]; then
echo "LOG_FILE_DIRECTORY $LOG_FILE_DIRECTORY not found, creating";
mkdir -p ${LOG_FILE_DIRECTORY}; fi

if [ -z "$TOTAL_GENERATIONS" ]; then
  echo "TOTAL_GENERATIONS not defined, this is how many generations deep the job is."; exit 1; fi

if [ -z "$NUM_STEPS" ]; then
  echo "NUM_STEPS not defined, this is how many workers run per generation."; exit 1; fi

if [ -z "$THREADS_PER_STEP" ]; then
  echo "THREADS_PER_STEP not defined, this is how many processors each step requires."; exit 1; fi

if [ -z "$STEP_WORKER_SCRIPT" ]; then
  echo "STEP_WORKER_SCRIPT not defined, this is what is run on every job step."; exit 1; fi

#End sanity tests

#Below here are the loops to generate the generational job files with
#The steps inside them

if [ "$(ls jobfiles/${JOB_NAME}_*.sh 2> /dev/null)" ]; then
  echo "Error, job files exist with this prefix";
  exit 1; fi

#Begin Generation Loop#
for GENERATION in $(seq 0 $(expr $TOTAL_GENERATIONS - 1)); do
  cat << EOF >> ${JOB_FILE_DIRECTORY}/${JOB_NAME}_${GENERATION}.sh
#!/bin/bash -l
#
#
# This is an example job file for a single core CPU bound program
# Note that all of the following statements below that begin
# with #SBATCH are actually commands to the SLURM scheduler.
# Please copy this file to your home directory and modify it
# to suit your needs.
#
# If you need any help, please email rc-help@rit.edu
#
# Name of the job
#SBATCH -J ${JOB_NAME}_${GENERATION}

# Standard out and Standard Error output files
#SBATCH -o ${LOG_FILE_DIRECTORY}/${JOB_NAME}_${GENERATION}.stdout

EOF

`
CHAPTER Q. Computing cluster submission scripts

```bash
#SBATCH -e ${LOG_FILE_DIRECTORY}/${JOB_NAME}_${GENERATION}.stderr
#SBATCH -t ${SLURM_WALLCLOCK}
#SBATCH --qos=${SLURM_QOS}
#SBATCH -p ${SLURM_PARTITION} -n ${NUM_STEPS} -c ${THREADS_PER_STEP}
#SBATCH --mem=${SLURM_MEMORY_REQ}

# A check to see if this script is managed by SLURM
/usr/bin/env | grep SLURM_JOB_ID
if [ "$?" != "0" ] ; then
  echo "Please run this script with 'sbatch <script-name>'"
  echo "Email rc-help@rit.edu if you have any questions."
  echo "Aborting."
else
EOF

#Begin Step Loop#
for STEPNUMBER in $(seq 0 $(expr ${NUM_STEPS} - 1)) ; do
  echo "srun -n 1 -c ${THREADS_PER_STEP} -o ${LOG_FILE_DIRECTORY}/${JOB_NAME}_${STEPNUMBER}.log ${STEP_WORKER_SCRIPT} ${GENERATION} ${STEPNUMBER} &" >> ${JOB_FILE_DIRECTORY}/${JOB_NAME}_${GENERATION}.sh
done

#End Step Loop#
cat << EOF >> ${JOB_FILE_DIRECTORY}/${JOB_NAME}_${GENERATION}.sh
#wait for above jobs to finish
wait

#release the next generation
if [ "$GENERATION" -lt "$TOTAL_GENERATIONS" ] ; then
  scontrol release \${squeue --noheader --format "%.i,%.j" | grep ",${JOB_NAME}_"$(expr ${GENERATION} + 1)" | cut -f 1 -d ,)
fi

fi
EOF

#Lets submit the new job file
```

sbatch --hold ${JOB_FILE_DIRECTORY}/${JOB_NAME}_${GENERATION}.sh

#End Generation Loop#
Appendix R

WASP Calibration Code

All calibration code for the WASP sensor were developed in IDL and can be run headless through any Unix or Linux based terminal.

R.1 WASP_CAL_FILEHANDLER

This procedure performs the file handling for the WASP calibration process. It assumes that inside a flight directory there are directories containing blackbody imagery as well as scene imagery. Additionally, a log file from the black body system, bbcal_log.csv, is also in the same directory. Using the image numbers and recorded temperatures associated with each file number, the handler determines the appropriate bracketing hot and cold blackbody images to use for each flight line to be calibrated in the data set. This process is what calls the main CAL_WASP_PREPROCESSING which performs the actual processing.

```idl
1 PRO WASP_CAL_FILEHANDLER, flightlinedir
2 CD, flightlinedir
3 SPAWN, 'pwd', calPath
4 tempreffile = FILE_WHICH(calPath, 'bbcal_log.csv')
5 CD, flightlinedir
6
7 ; open the BB log file and read in data
8 nlines = FILE_LINES(tempreffile)
9 tempdata = FLTARR(10, nlines)
10 CD, flightlinedir
11
12 OPENR, lun, tempreffile, /GET_LUN
13 READF, lun, tempdata
14 CLOSE, lun
```
FREE_LUN, lun

midTemp = MEAN(tempdata[3, *])

hotfixed = FIX(tempdata[0, WHERE(tempdata[3, *] GT midTemp)])

hotimgNumString = STRTRIM(STRING(hotfixed), 2)

IF hotsmallNumLocs[0] NE -1 THEN hotimgNumString[hotsmallNumLocs] = '0' + hotimgNumString[hotsmallNumLocs]

coldfixed = FIX(tempdata[0, WHERE(tempdata[3, *] LT midTemp)])
coldimgNumString = STRTRIM(STRING(coldfixed), 2)
coldsmallNumLocs = WHERE(STRLEN(coldimgNumString) LE 2)

IF coldsmallNumLocs[0] NE -1 THEN coldimgNumString[coldsmallNumLocs] = '0' + coldimgNumString[coldsmallNumLocs]

hotImgNames = REFORM("LWIR" + hotimgNumString + ".img")

print, "The Hot images are: " +hotImgNames

coldImgNames = REFORM("LWIR" + coldimgNumString + ".img")

print, "The Cold images are: " + coldImgNames

numHot = N_ELEMENTS(hotImgNames)
numCold = N_ELEMENTS(coldImgNames)
hotDirArray = STRARR(2, 2)
coldDirArray = STRARR(2, 2)
calDirArray = STRARR(2, 2)

FOR curHot=0, numHot-1 DO BEGIN
  curPath = FILE_SEARCH(flightlinedir, hotImgNames[curHot])
curPath = FILE_DIRNAME(curPath)
curPath = FILE_BASENAME(curPath)
numDirs = N_ELEMENTS(hotDirArray[0, *])
  IF hotDirArray[0, numDirs-1] EQ curPath THEN BEGIN
    CONTINUE
  ENDIF ELSE BEGIN
    pathComps = STRSPLIT(curPath, "-", /EXTRACT)
timeString = STRING(pathComps[3]) + STRING(pathComps[4]) + STRING(pathComps[5])
  hotDirArray = [[hotDirArray], [curPath, timeString]]
  ENDELSE
  ENDFOR

FOR curCold=0, numCold-1 DO BEGIN
  curPath = FILE_SEARCH(flightlinedir, coldImgNames[curCold])
curPath = FILE_DIRNAME(curPath)
curPath = FILE_BASENAME(curPath)
numDirs = N_ELEMENTS(coldDirArray[0, *])
  IF coldDirArray[0, numDirs-1] EQ curPath THEN BEGIN
    CONTINUE
  ENDIF ELSE BEGIN
    pathComps = STRSPLIT(curPath, "-", /EXTRACT)
timeString = STRING(pathComps[3]) + STRING(pathComps[4]) + STRING(pathComps[5])
  coldDirArray = [[coldDirArray], [curPath, timeString]]
  ENDELSE
  ENDFOR

SPAWN, 'ls', allDirs, /NOSHELL
numDirs = N_ELEMENTS(allDirs)
calCount = 0
numCalCountDirs = numDirs - (N_ELEMENTS(coldDirArray[0, *]) - 2) - (N_ELEMENTS(hotDirArray[0, *]) - 2)
tempdataPTR = PTR_NEW(tempdata, /ALLOCATE_HEAP, /NO_COPY)
tempdata = 0

FOR curDir=0, numDirs-1 DO BEGIN
  coldMatch = WHERE(allDirs[curDir] EQ coldDirArray[0, *])
  hotMatch = WHERE(allDirs[curDir] EQ hotDirArray[0, *])
  numCalDirs = N_ELEMENTS(calDirArray[0, *])
  IF (coldMatch[0] NE -1) OR (hotMatch[0] NE -1) THEN BEGIN
    CONTINUE
  ENDIF ELSE BEGIN
    calCount++
    CD, alldirs[curdir]
    SPAWN, 'ls', filecount
    IF N_ELEMENTS(filecount) LE 6 THEN BEGIN
      CD, '..'
      CONTINUE
    ENDIF
    CD, '..'
    pathComps = STRSPLIT(allDirs[curDir], "-", /EXTRACT)
    timeString = STRING(pathComps[3]) + STRING(pathComps[4]) + STRING(pathComps[5])
    calDirArray = [[calDirArray], [allDirs[curDir], timeString]]
    diffHotArray = ABS(FLOAT(calDirArray[1, numCalDirs])-FLOAT(hotDirArray[1, 2:*]))
    minHotDiff = MIN(ABS(FLOAT(calDirArray[1, numCalDirs]) - FLOAT(hotDirArray[1, 2:*])))
    minHotDiffLoc = WHERE(diffHotArray EQ minHotDiff)
    closestHotTime = hotDirArray[1, minHotDiffLoc+2]
    diffColdArray = ABS(FLOAT(calDirArray[1, numCalDirs])-FLOAT(coldDirArray[1, 2:*]))
    minColdDiff = MIN(ABS(FLOAT(calDirArray[1, numCalDirs]) - FLOAT(coldDirArray[1, 2:*])))
    minColdDiffLoc = WHERE(diffColdArray EQ minColdDiff)
    closestColdTime = coldDirArray[1, minColdDiffLoc+2]
  coldDir = coldDirArray[0, WHERE(coldDirArray[1, *] EQ closestColdTime[0])]
  hotDir = hotDirArray[0, WHERE(hotDirArray[1, *] EQ closestHotTime[0])]
  PRINT, "Beginning to calibrate LWIR directory: " + allDirs[curDir] + $ " [" +STRCOMPRESS(STRING(calCount), /REMOVE_ALL) + $ "/" +STRCOMPRESS(STRING(numCalCountDirs), /REMOVE_ALL) + "]" print, "Using " + coldDir[0] + " and " + HotDir[0] + " for cold and hot calibration images"
status = CAL_WASP_PREPROCESSING(flightLineDir+coldDir[0], flightLineDir+hotDir[0], $ flightLineDir+allDirs[curDir], tempdataPTR, 'LWIR')
PRINT, "Finished calibrating LWIR in directory: " + allDirs[curDir] + $ "/" +STRCOMPRESS(STRING(numCalCountDirs), /REMOVE_ALL) + "]"
PRINT, "Beginning to calibrate MWIR directory: " + allDirs[curDir] + " "$ \\
(" +STRCOMPRESS(STRING(calCount), /REMOVE_ALL) + "$ \\
"/" +STRCOMPRESS(STRING(numCalCountDirs), /REMOVE_ALL) + "]"
status = CAL_WASP_PREPROCESSING(flightLineDir+coldDir[0], flightLineDir+hotDir[0], \\
flightLineDir+allDirs[curDir], tempdataPTR, 'MWIR')
PRINT, "Finished calibrating MWIR in directory: " + allDirs[curDir] + "$ \\
(" +STRCOMPRESS(STRING(calCount), /REMOVE_ALL) + "$ \\
"/" +STRCOMPRESS(STRING(numCalCountDirs), /REMOVE_ALL) + "]"
ENDELSE
ENDIF
PTR_FREE, tempdataPTR
END

R.2 CAL_WASP_PREPROCESSING

; + NAME:
; CAL_WASP_PREPROCESS
; PURPOSE:
; This procedure uses the logged BB temperatures and their corresponding timestamps 
; to choose the images 
; to calibrate the un-calibrated WASP imagery from a single flight line. The set of 
; both hot and cold images 
; are averaged to create a single hot and cold image to represent the bracketing BB 
; images for a given flight 
; line. The measured temperature of the blackbody for those corresponding images is 
; also averaged from the 
; log file. Using Planck’s equation, both blackbody image sets, and the recorded 
; temperatures, the raw WASP 
; imagery contained within each flight line is calibrated out radiance units using 
; Wasp_THERM_COMMAND one image 
; at a time. The final produced image cubes are a four-banded floating point data 
; cube with the same dimensions 
; as the original raw imagery. The first band is the raw data, the second band is 
; the gain applied to the raw 
; image, the third band is the bias applied to the raw image, and the fourth band is 
; the final radiance image.

; CALLING SEQUENCE:
; CAL_WASP_PREPROCESS
; INPUT ARGUMENT:
; None
; RETURNS:
; None
; INTERNAL CALLS:
; OPEN_ENVI_IMG_COMMAND
FUNCTION CAL_WASP_PREPROCESSING, dirColdImages, dirHotImages, dirImages, tempdataPTR, band

COMMON calibrationBlock, coldavg, hotavg, hotStdDevArr, coldStdDevArr

IF ( !VERSION.OS_FAMILY EQ 'Windows' ) THEN delimit='\' ELSE delimit='/'

workingpath = FILE_DIRNAME(dirImages)

searchPattern = STRCOMPRESS(band + '*.img', /REMOVE_ALL)
calfilelist = FILE_SEARCH(dirImages, searchPattern)
coldimagelist = FILE_SEARCH(dirColdImages, searchPattern)
hotimagelist = FILE_SEARCH(dirHotImages, searchPattern)

rawDirectory = FILE_DIRNAME(calfilelist[0])
calDirectory = FILE_DIRNAME(dirColdImages)
calDirName = STRCOMPRESS(STRLOWCASE(band)+'-proc', /REMOVE_ALL)

calDirectory = calDirectory + delimit + calDirName
FILE_MKDIR, calDirectory

coldcount = N_ELEMENTS(coldimagelist)
hotcount = N_ELEMENTS(hotimagelist)
calcount = N_ELEMENTS(calfilelist)

coldsum = FLOAT(0)
hotsum = FLOAT(0)

coldthermsum = FLOAT(0)
hotthermsum = FLOAT(0)
sampleImg = READ_ENVI_IMAGE(coldimagelist[0])
sampleImg = 0
numpix = size[0]*size[1]
xsize = size[0]
ysize = size[1]
coldBBcube = FLTARR(coldcount, xsize, ysize)
hotBBcube = FLTARR(hotcount, xsize, ysize)
tempdata = *tempdataPTR

IF band EQ 'LWIR' THEN dataCol=6 ELSE dataCol=4

; average each of the cold black body sequence images - also average the physical
thermistor measurement
FOR i=0, coldcount-1 DO BEGIN
coldBBcube[i, *, *) = READ_ENVI_IMAGE(coldimagelist[i])
coldsum = FLOAT(coldsum) + coldBBcube[i, *, *)
coldnum = STRSPLIT(FILE_BASENAME(coldimagelist[i],'.img'),band,/EXTRACT)
coldthermsum = FLOAT(coldthermsum) + tempdata[datacol, WHERE(tempdata[0, *) EQ
coldnum[0])]
ENDFOR

coldavg = REFORM(FLOAT(coldsum / coldcount), numpix)
coldthermavg = FLOAT(coldthermsum / coldcount)

coldBBcube = 0

; average each of the hot black body sequence images - also average the physical
thermistor measurement
FOR i=0, hotcount-1 DO BEGIN
hotBBcube[i, *, *) = READ_ENVI_IMAGE(hotimagelist[i])
hotsum = FLOAT(hotsum) + hotBBcube[i, *, *)
hotnum = STRSPLIT(FILE_BASENAME(hotimagelist[i],'.img'),band,/EXTRACT)
hotthermsum = FLOAT(hotthermsum) + tempdata[datacol, WHERE(tempdata[0, *) EQ
hotnum[0])]
ENDFOR

hotBBcube = 0
coldsum = 0
hotsum = 0
coldthermsum = 0
hotthermsum = 0
coldavgcube = 0
hotavgcube = 0
colddemeanedcube = 0
hotdemeanedcube = 0
hotvertones = 0
coldvertones = 0

hotavg = \text{REFORM}(\text{hotavg}, xsize, ysize)
coldavg = \text{REFORM}(\text{coldavg}, xsize, ysize)

; determine size of image window
windowSize = 64.0

; determine how many segments can be created
numColBoxes = xsize / windowSize
numRowBoxes = ysize / windowSize

; create empty array to hold head pixel mask
deadPixMask = \text{INTARR} (xsize, ysize)

; create empty columns to append to final image - handles two leftmost empty columns in every WASP image
coldEmptyCols = \text{FLTARR}(2, windowSize)
hotEmptyCols = \text{FLTARR}(2, windowSize)

\text{FOR} \ i = 0, \text{numColBoxes}-1 \ \text{DO BEGIN}
\text{FOR} \ j = 0, \text{numRowBoxes}-1 \ \text{DO BEGIN}

; select subset of average hot and cold blackbody images
hotavgsub = \text{hotavg}[\text{i} \text{\cdot windowSize}:(\text{i} \text{\cdot windowSize} + \text{windowSize})-1, \text{(j} \text{\cdot windowSize}:(\text{j} \text{\cdot windowSize} + \text{windowSize})-1]
coldavgsub = \text{coldavg}[\text{(i} \text{\cdot windowSize}:(\text{i} \text{\cdot windowSize} + \text{windowSize})-1, \text{(j} \text{\cdot windowSize}:(\text{j} \text{\cdot windowSize} + \text{windowSize})-1]
subdims = \text{SIZE}(\text{coldavgsub}, /\text{DIMENSIONS})
deadPixMasksub = \text{INTARR}(\text{subdims}[0], \text{subdims}[1])

; calculate histogram threshold for the top and bottom 1% of digital counts
hotThresh = 0.02 \cdot \text{MAX}(\text{hotavgsub})
coldThresh = 0.02 \cdot \text{MAX}(\text{coldavgsub})

; calculate maximum and minimum digital counts allowable
hotTop = \text{MAX}(\text{hotavgsub}) - \text{hotThresh}
hotBottom = \text{MIN}(\text{hotavgsub}) + \text{hotThresh}
coldTop = \text{MAX}(\text{coldavgsub}) - \text{coldThresh}
coldBottom = \text{MIN}(\text{coldavgsub}) + \text{coldThresh}

; determine location of all pixels above and below the thresholds (i.e. dead pixels)
hotOutLocs = \text{WHERE}(\text{hotavgsub} \ \text{GE} \ \text{hotTop}) \ \text{OR} \ (\text{hotavgsub} \ \text{LE} \ \text{hotbottom}), \text{hotOutCount}
coldOutLocs = \text{WHERE}(\text{coldavgsub} \ \text{GE} \ \text{coldTop}) \ \text{OR} \ (\text{coldavgsub} \ \text{LE} \ \text{coldBottom}), \text{coldOutCount}
lowThresh = 0.0025
highThresh = 0.9975
CDFhot = TOTAL( HISTOGRAM(hotavgsub, LOCATIONS=hotLocs)/FLOAT(N_ELEMENTS(hotavgsub)), /CUMULATIVE)
CDFcold = TOTAL( HISTOGRAM(coldavgsub, LOCATIONS=coldLocs)/FLOAT(N_ELEMENTS(coldavgsub)), /CUMULATIVE)
minHotDCs = WHERE( ABS( CDFhot - lowThresh ) EQ MIN( ABS( CDFhot - lowThresh )), numHotMin)
minColdDCs = WHERE( ABS( CDFcold - lowThresh ) EQ MIN( ABS( CDFcold - lowThresh )), numColdMin)
maxHotDCs = WHERE( ABS( CDFhot - highThresh ) EQ MIN( ABS( CDFhot - highThresh )), numHotMax)
maxColdDCs = WHERE( ABS( CDFcold - highThresh ) EQ MIN( ABS( CDFcold - highThresh )), numColdMax)
deadPixMasksub[WHERE(hotavgsub GE hotLocs[maxHotDCs[0]])] = 1.0
deadPixMasksub[WHERE(hotavgsub LE hotLocs[minHotDCs[numHotMin-1]])] = 1.0
deadPixMasksub[WHERE(coldavgsub GE coldLocs[maxColdDCs[0]])] = 1.0
deadPixMasksub[WHERE(coldavgsub LE coldLocs[minColdDCs[numColdMin-1]])] = 1.0

; create a dead pixel mask and set each dead pixel location to 1
; deadPixMasksub[hotOutLocs] = 1
; deadPixMasksub[coldOutLocs] = 1

; fill total dead pixel mask with dead pixels found in current subset
deadPixMasksub[WHERE(hotavgsub GE hotLocs[maxHotDCs[0]])] = 1.0
deadPixMasksub[WHERE(hotavgsub LE hotLocs[minHotDCs[numHotMin-1]])] = 1.0
deadPixMasksub[WHERE(coldavgsub GE coldLocs[maxColdDCs[0]])] = 1.0
deadPixMasksub[WHERE(coldavgsub LE coldLocs[minColdDCs[numColdMin-1]])] = 1.0

; fill in hot and cold blackbody images with dead pixel-filled subset
hotavg[(i*windowSize):(i*windowSize+windowSize)-1, (j*windowSize):(j*windowSize+windowSize)-1] = hotavgsub
coldavg[(i*windowSize):(i*windowSize+windowSize)-1, (j*windowSize):(j*windowSize+windowSize)-1] = coldavgsub

ENDFOR
ENDFOR
deadLocs = WHERE(deadPixMask EQ 1)
umDead = N_ELEMENTS(deadLocs)
smoothedHot = RESAMP_DEADPIX(deadPixMask, /HOT)
smoothedCold = RESAMP_DEADPIX(deadPixMask, /COLD)

; replace only known dead pixels with calculated averages
hotavg[deadLocs] = smoothedHot[deadLocs]
coldavg[deadLocs] = smoothedCold[deadLocs]

smoothedHot = 0
smoothedCold = 0
BBData = FLTARR(7, xsize, ysize)
BBData[0, *, *] = hotoriginal
BBData[1, *, *] = coldoriginal
BBData[2, *, *] = deadPixMask*255.0
BBData[3, *, *] = hotavg
BBData[4, *, *] = coldavg

; Convert each raw image tile into radiance units. Output is a 4-banded image cube
; containing the raw tile,
; the calculated gain for each pixel, the calculated bias for each pixel, and the
; produced radiance image.
; Write the image cube into the "lwir-proc" directory.
FOR i=0, calcount-1 DO BEGIN
  preProcessedCube = WASP_THERM_COMMAND( calfilelist[i], coldthermavg[0],
  hotthermavg[0], deadPixMask, band )
  rawBaseName = FILE_BASENAME(calfilelist[i], '.img')
  cubeName = calDirectory + delim + rawBaseName + '_PreProcCube.tif'
  WRITE_TIFF, cubeName, preProcessedCube, /FLOAT
  PRINT, "Done with image: " + rawBaseName + '_PreProcCube.tif'
  preProcessedCube = 0
ENDFOR

coldavg = 0
hotavg = 0
hotStdDevArr = 0
coldStdDevArr = 0
RETURN, 1
END

R.3 WASP_THERM_COMMAND

1 ;
2 ; NAME:
3 ; WASP_THERM_COMMAND
4 ;
5 ; PURPOSE:
6 ; The function WASP_THERM_COMMAND is used to generate the four banded image cube
7 ; comprised of the calibrated radiance image,
8 ; the associated gains and biases used to generate the calibration, and the original
9 ; raw image tiles.
10 ;
11 ; CALLING SEQUENCE:
12 ; PREPROCESSEDCUBE = WASP_THERM_COMMAND( rawImg, coldBBImg, coldT, hotBBImg, hotT )
13 ;
14 ; INPUT ARGUMENT:
15 ; rawImg = array containing spatial information of raw imagery to be calibrated
16 ; coldBBImg = array containing spatial information of raw cold blackbody associated
17 ; with the flightline
18 ; coldT = the measured cold temperature of the cold blackbody at the time the image
19 ; was acquired
20 ; hotBBImg = array containing spatial information of raw hot blackbody associated
21 ; with the flightline
22 ; hotT = the measured hot temperature of the hot blackbody at the time the image was
23 ; acquired
24 ;
25 ; RETURNS:
26 ; PREPROCESSEDCUBE = four banded imaged containing the original raw image, associated
27 ; gain mask, associated
28 ; bias mask, and processed radiance image
FUNCTION WASP_THERM_COMMAND, rawImgfile, coldT, hotT, deadPixMask, band

COMMON calibrationBlock, coldavg, hotavg, hotStdDevArr, coldStdDevArr

; Open raw imagery and gather dimensions information
rawImg = READ_ENVI_IMAGE(rawImgfile)
rawDIMS = SIZE(rawImg, /DIMENSIONS)
rawImg = FLOAT(rawImg)
deadPixLocs = WHERE(deadPixMask EQ 1)
smoothedRaw = RESAMP_DEADPIX(deadPixMask, IMG=rawImg)

; replace dead pixels with interpolated averages
rawImg[deadPixLocs] = smoothedRaw[deadPixLocs]

; Compute the radiance emitted from the hot and cold blackbodies. Use radiance values to derive linear fit
radArr = COMPUTE_RADIANCE(hotT, coldT, band)
mask_cube = COMPUTE_SLOPE_INT(radArr[0], radArr[1])
error_cube = COMPUTE_STD_LININTERP()

; Apply calculated gain and bias to the raw imagery pixel-by-pixel
slope_img = rawImg * mask_cube[1, *, *]
calRadImg = slope_img + mask_cube[0, *, *]

; Build 6-banded image cube
preProcessedCube = FLTARR(6, rawdims[0], rawdims[1] )
preProcessedCube[0, *, *] = rawImg
preProcessedCube[1, *, *] = mask_cube[1, *, *]
R.4. COMPUTE_RADIANCE

```
preProcessedCube[2, *] = mask_cube[0, *]
preProcessedCube[3, *] = calRadImg
preProcessedCube[4, *] = error_cube[1, *]
preProcessedCube[5, *] = error_cube[0, *]
RETURN, preProcessedCube
END
```

**R.4 COMPUTE_RADIANCE**

```plaintext
; NAME: COMPUTE_RADIANCE
; PURPOSE: The function COMPUTE_RADIANCE takes in the temperatures of the hot and cold blackbody that was used for the calibration images and computes the blackbody radiance via the Planck function. Then it takes into account the response of WASP’s LWIR sensor and applies the sensor response function to the blackbody radiance. Finally it integrates under the radiance curve to compute the total radiance in units W/m^2/sr/micron.

; CALLING SEQUENCE:
RAARR = COMPUTE_RADIANCE(hotT, coldT)

; INPUT ARGUMENT:
hotT = temperature of the hot blackbody
coldT = temperature of the cold blackbody

; RETURNS:
RAARR = array of calculated total radiance values, [TotalHotRad, TotalColdRad]

; INTERNAL CALLS:
None

; OPTIONAL OUTPUT ARGUMENT:
None

; OPTIONAL INPUT KEYWORD:
None

; NOTES:
None

; WARNING:
Currently the sensitivity data hard-coded into this function are for the WASP LWIR sensor as of 10/8/2009. To apply the calibration routine to another framing device this data array either needs to be replaced with the new sensor’s sensitivity. In addition, the emissivity data hard-coded in are for the blackbody material mounted on the WASP sensor as of 10/8/2009. This array would have to be changed if the blackbody material is replaced.
```
CHAPTER R. WASP Calibration Code

; REVISION HISTORY:
; Written M.V. Casterline, 10/8/09, RIT-DIRS

FUNCTION COMPUTE_RADIANCE, hotTemp, coldTemp, band

; WASP outputs a single band image that has integrated energy from 8 - 9.2 microns.
; Sixty wavelengths and their corresponding sensitivity value from the WASP LWIR response
; curve were put into a comma delimited text file. This text file is called ‘
; wasp_lwir_response.csv’
; and could be used in the READ_ASCII command. Currently this data as well as the BB
; emissivity
; is hard-coded in below.

IF band EQ 'MWIR' THEN BEGIN
  sensitivityArr= [[3.0, 0.56], $ 
  [3.0833, 0.6], $ 
  [3.1666, 0.625], $ 
  [3.2499, 0.65], $ 
  [3.3333, 0.675], $ 
  [3.4165, 0.7], $ 
  [3.5, 0.725], $ 
  [3.5833, 0.775], $ 
  [3.6666, 0.8], $ 
  [3.7499, 0.85], $ 
  [3.8333, 0.8], $ 
  [3.9166, 0.83], $ 
  [4.0, 0.85], $ 
  [4.0833, 0.86], $ 
  [4.1666, 0.875], $ 
  [4.2499, 0.925], $ 
  [4.3333, 0.92], $ 
  [4.4166, 0.875], $ 
  [4.5, 0.925], $ 
  [4.5833, 0.925], $ 
  [4.6666, 0.93], $ 
  [4.7499, 0.95], $ 
  [4.8333, 0.97], $ 
  [4.9166, 0.96], $ 
  [5.0, 0.95], $ 
  [5.0833, 0.98], $ 
  [5.1666, 1.0], $ 
  [5.2499, 0.95], $ 
  [5.3333, 0.85], $ 
  [5.4166, 0.775], $ 
  [5.5, 0.28], $ 
  [5.5833, 0.04], $ 
  [5.6666, 0.01], $ 
  [5.7499, 0.05]]

  emissivity = [0.961, 0.9609, 0.9605, 0.9612, 0.9629, $ 
  0.9601, 0.9596, 0.9592, 0.9588, 0.9592, $ 
  0.9595, 0.9594, 0.959, 0.9593, 0.9593, $ 
  0.9574, 9.9588, 0.9591, 0.9591, 0.9594, $ 
  0.9574, 0.9588, 0.9591, 0.9591, 0.9594, $ 
  0.9574, 0.9588, 0.9591, 0.9591, 0.9594, $]
R.4. COMPUTE_RADIANCE

92 0.9592, 0.9591, 0.9595, 0.96, 0.96, 0.96, $
93 0.9606, 0.9609, 0.9616, 0.9621, 0.9634, $
94 0.965, 0.9656$ 
95 ENDIF ELSE BEGIN
96 sensitivityArr = [[8, 0.3775], $
97 [8.02, 0.3725], $
98 [8.04, 0.37], $
99 [8.06, 0.3675], $
100 [8.08, 0.37], $
101 [8.1, 0.37], $
102 [8.12, 0.3675], $
103 [8.14, 0.3675], $
104 [8.16, 0.3725], $
105 [8.18, 0.39], $
106 [8.2, 0.4075], $
107 [8.22, 0.4425], $
108 [8.24, 0.485], $
109 [8.26, 0.5125], $
110 [8.28, 0.56], $
111 [8.3, 0.62], $
112 [8.32, 0.65], $
113 [8.34, 0.6525], $
114 [8.36, 0.65], $
115 [8.38, 0.645], $
116 [8.4, 0.65], $
117 [8.42, 0.66], $
118 [8.44, 0.695], $
119 [8.46, 0.74], $
120 [8.48, 0.805], $
121 [8.5, 0.8475], $
122 [8.52, 0.89], $
123 [8.54, 0.925], $
124 [8.56, 0.9625], $
125 [8.58, 0.9875], $
126 [8.6, 1], $
127 [8.62, 0.9975], $
128 [8.64, 0.99], $
129 [8.66, 0.9775], $
130 [8.68, 0.9625], $
131 [8.7, 0.9575], $
132 [8.72, 0.9575], $
133 [8.74, 0.96], $
134 [8.76, 0.9575], $
135 [8.78, 0.9525], $
136 [8.8, 0.945], $
137 [8.82, 0.915], $
138 [8.84, 0.89], $
139 [8.86, 0.8625], $
140 [8.88, 0.84], $
141 [8.9, 0.82], $
142 [8.92, 0.7925], $
143 [8.94, 0.7625], $
144 [8.96, 0.7225], $
145 [8.98, 0.6825], $
146 [9.0, 0.645], $
147 [9.02, 0.61], $
148 [9.04, 0.589], $
149 $
CHAPTER R. WASP Calibration Code

\[
\text{emissivity} = [0.9778, 0.9778, 0.9773, 0.9763, 0.9759, 0.9754, 0.9742, 0.9739, 0.9742, 0.9744, 0.9749, 0.9754, 0.9758, 0.9764, 0.9773, 0.9781, 0.9784, 0.9784, 0.9782, 0.9776, 0.9763, 0.9755, 0.974, 0.9731, 0.9691, 0.9681, 0.9669, 0.9663, 0.9648, 0.9635, 0.9621, 0.9596, 0.9585, 0.9571, 0.9567, 0.9562, 0.9552, 0.9548, 0.9542, 0.9532, 0.9529, 0.9527, 0.9533, 0.9535, 0.9532, 0.953, 0.9527, 0.9525, 0.9518, 0.9518, 0.9519, 0.9523, 0.9528, 0.9529, 0.953, 0.9526, 0.9526, 0.95625, 0.9532]
\]

; Section into wavelengths, sensitivity, and emissivity
wavelength = sensitivityArr[0, *]
sensitivity = sensitivityArr[1, *]
emissivity = TRANSPOSE(emissivity)

; Convert blackbody temperatures that are in degrees C to Kelvin
tempC = coldTemp + 273.15
tempH = hotTemp + 273.15

; Compute radiance values for hotTemp using wavelength array generated from response curve chart
; should be in units of W/m²/sr/micron
c1 = 3.74515e8
c2 = 1.43879e4
w = FLOAT(wavelength)
tH = FLOAT(tempH)
tC = FLOAT(tempC)
radH = c1/((PI * w^5) * ( EXP(c2/(w * tH)) - 1))

; Multiply each radiance value by BB emissivity and sensor response
radianceHot = radH * sensitivity * emissivity

; Need to get total radiance, so the radiance curve from the previous step is integrated using
; built in function INT_TABULATED
totalRadianceHot = INT_TABULATED(wavelength, radianceHot)

; Now repeat for cold temp
radC = c1/((PI * w^5) * ( EXP(c2/(w * tC)) - 1))

; Multiply each radiance value by BB emissivity and sensor response
radianceCold = radC * sensitivity * emissivity
R.5 COMPUTE_SLOPE_INT

204; Calculate total radiance
205 totalRadianceCold = INT_TABULATED(wavelength, radianceCold)
206
207RETURN, [[totalRadianceHot],[totalRadianceCold]]
208
209END

R.5 COMPUTE_SLOPE_INT

1 ;
2 ; NAME:
3 ; COMPUTE_SLOPE_INT
4 ;
5 ; PURPOSE:
6 ; The function COMPUTE_SLOPE_INT takes in the images of the hot and cold blackbody and also
7 ; the radiance values from the function COMPUTE_RADIANCE and calculates the slope (gain) and
8 ; y-intercept (bias) at each pixel in the cold blackbody image. The function returns a
9 ; slope mask and a bias mask that will be used in the main procedure to calibrate a raw
10 ; WASP LWIR image.
11 ;
12 ; CALLING SEQUENCE:
13 ; MASK_CUBE = COMPUTE_SLOPE_INT(coldBBImg, hotBBImg, coldRadiance, hotRadiance)
14 ;
15 ; INPUT ARGUMENT:
16 ; coldBBImg = average cold blackbody image
17 ; hotBBImg = average hot blackbody image
18 ; coldRadiance = calculated total radiance emitted from the cold blackbody
19 ; hotRadiance = calculated total radiance emitted from the hot blackbody
20 ;
21 ; RETURNS:
22 ; MASK_CUBE = Two-banded array with the same dimensions as input BB images. First
23 ; band contains the pixel-by-pixel gain to be applied to the raw imagery,
24 ; while the second band contains the pixel-by-pixel bias to be applied to the raw
25 ; imagery.
26 ;
27 ; INTERNAL CALLS:
28 ; None
29 ;
30 ; OPTIONAL OUTPUT ARGUMENT:
31 ; None
32 ;
33 ; OPTIONAL INPUT KEYWORD:
34 ; None
35 ;
36 ; NOTES:
37 ; None
38 ;
39 ; WARNING:
40 ; None
41 ;
FUNCTION COMPUTE_SLOPE_INT, radiance_hot, radiance_cold

COMMON calibrationBlock, coldavg, hotavg, hotStdDevArr, coldStdDevArr
; Compute the numerator of the slope equation by subtracting radiance_cold from radiance_hot
rise = radiance_hot - radiance_cold
; Compute the denominator of the slope equation by subtracting the cold black body image from the hot blackbody image
diffImage = FLOAT(hotavg) - FLOAT(coldavg)
; Create slope mask by computing slope equation
index = WHERE( diffImage EQ 0, findcount )
IF (findcount NE 0) THEN BEGIN
  diffImage[index] = diffImage[index] + 0.001
ENDIF
slopeMask = FLOAT((rise / diffImage))
; Compute the y-intercept (the bias) using cold blackbody info
; ColdRadiance = (slopeMask * DC_coldBB) + bias
; bias = ColdRadiance - (slopeMask * DC_coldBB)
biasMask = radiance_cold - (slopeMask * coldavg)
dims = SIZE(biasMask, /DIMENSIONS)
maskcube = FLTARR(2, dims[0], dims[1])
maskcube[0, *, *] = biasMask
maskcube[1, *, *] = slopeMask
RETURN, maskcube
END

R.6 COMPUTE_STD_LININTERP

; NAME:
; COMPUTE_STD_LININTERP
; PURPOSE:
; CALLING SEQUENCE:
; ERROR_CUBE = COMPUTE_STD_LININTERP( hotStdDevArr, coldStdDevArr )
; INPUT ARGUMENT:
; RETURNS:
; ERROR_CUBE = 
; INTERNAL CALLS:
; None
FUNCTION COMPUTE_STD_LININTERP

COMMON calibrationBlock, coldavg, hotavg, hotStdDevArr, coldStdDevArr

; Compute the numerator of the slope equation by subtracting coldStdDevArr from hotStdDevArr
rise = hotStdDevArr - coldStdDevArr

; Compute the denominator of the slope equation by subtracting the cold black body image from the hot blackbody image
diffImage = FLOAT(hotavg) - FLOAT(coldavg)

; Create slope mask by computing slope equation
index = WHERE( diffImage EQ 0, findcount )

IF (findcount NE 0) THEN BEGIN
   diffImage[index] = diffImage[index] + 0.001
ENDIF

slopeMask = FLOAT((rise / diffImage))

; Compute the y-intercept (the bias) using cold blackbody info
bias = ColdStdDevArr - (slopeMask * DC_coldBB)

biasMask = coldStdDevArr - (slopeMask * coldavg)

dims = SIZE(biasMask, /DIMENSIONS)
errorcube = FLTARR(2, dims[0], dims[1])

errorcube[0, *, *] = biasMask
errorcube[1, *, *] = slopeMask
RETURN, errorcube
END

FUNCTION RESAMP_DEADPIX, deadPixSub, IMG=img, HOT=hot, COLD=cold

COMMON calibrationBlock, coldavg, hotavg, hotStdDevArr, coldStdDevArr
IF ARG_PRESENT(img) THEN BEGIN
  ImgSub = img
ENDIF ELSE BEGIN
  IF KEYWORD_SET(hot) THEN imgSub=hotavg
  IF KEYWORD_SET(cold) THEN imgSub=coldavg
ENDELSE
s = SIZE(ImgSub)

;Form the x and y grids (pixel locations)
cols = s[1]
rows = s[2]
xgv=((FINDGEN(ncols)-(ncols-1)/2.)) ; X grid vector
ygv=((FINDGEN(nrows)-(nrows-1)/2.)) ; Y grid vector
xcoord=REPLICATE(1,nrows)#xgv ; Grid of x values
ycoord=REPLICATE(1,ncols)#ygv ; Grid of y values

; find 'bad' pixels
badInd = WHERE(deadPixSub EQ 1)
goodInd = WHERE(deadPixSub NE 1)

; locations of good values
xcoordGood = xcoord[goodInd]
ycoordGood = ycoord[goodInd]
zGood = imgSub[goodInd]

; locations to interpolate
xcoordBad = xcoord[badInd]
ycoordBad = ycoord[badInd]

TRIANGULATE, xcoordGood, ycoordGood, tr
methodN = "Linear"
badInterp = GRIDDATA(xcoordGood,ycoordGood,zGood,XOUT=xcoordBad,YOUT=ycoordBad,
                      METHOD=methodN,TRIANGLES = tr)

; put the interpolated values back into the image and display
imFixed = imgSub
imFixed[badInd] = badInterp
RETURN, imFixed
END

R.8 READ_ENVI_HEADER

;+ ; :NAME: READ_ENVI_HEADER ; ; ; :PURPOSE: This function serves to read an ENVI header directly into ; an IDL program. ; ; ; :CATEGORY: Image Processing.
; :CALLING SEQUENCE:
; Result = READ_ENVI_HEADER( filename )

; :INPUTS:
; filename:
; The ENVI header filename to be read.

; :KEYWORD PARAMETERs:
; NONE

; :RETURN VALUE:
; A structure containing the ENVI header information from the
; provided file. If any error is encountered during this process,
; the scalar -1 will be returned.

; :SIDE EFFECTs:
; NONE

; :MODIFICATION HISTORY:
; Written by: Carl Salvaggio
; July, 2009 Original code
; December, 2009 Header now includes the byte order

; DISCLAIMER:
; This source code is provided "as is" and without warranties as to performance
; or merchantability. The author and/or distributors of this source code may
; have made statements about this source code. Any such statements do not
; constitute warranties and shall not be relied on by the user in deciding
; whether to use this source code.

; This source code is provided without any express or implied warranties
; whatsoever. Because of the diversity of conditions and hardware under which
; this source code may be used, no warranty of fitness for a particular purpose
; is offered. The user is advised to test the source code thoroughly before
; relying on it. The user must assume the entire risk of using the source code.

FUNCTION READ_ENVI_HEADER, filename

; Define the header structure

; header = [ description: "", $ samples: -1L, $ lines: -1L, $ bands: -1L, $ header_offset: -1L, $ file_type: "", $ data_type: -1L, $ interleave: "", $ sensor_type: "", $ byte_order: -1L, $ wavelength_units: "", $ z_plot_range: [-1D, -1D], $ z_plot_titles: ["", ""], $ band_names: PTR_NEW(), $ wavelength: PTR_NEW() $
OPENR, lun, filename, /GET_LUN

OPENR, lun, filename, /GET_LUN

str = ""
READF, lun, str
str = STRTRIM( STRCOMPRESS( str ), 2 )
IF( str NE "ENVI" ) THEN BEGIN
    MESSAGE, "ENVI header file has an invalid format", /CONTINUE
    RETURN, -1
ENDIF

WHILE NOT EOF( lun ) DO BEGIN
    READF, lun, str

    IF( STRLEN( str ) GT 0 ) THEN BEGIN
        equalPosition = STRPOS( str, "=" )
        name = STRTRIM( STRMID( str, 0, equalPosition ), 2 )
        value = STRTRIM( STRMID( str, equalPosition+1 ), 2 )
        multilineValue = STRTRIM( value, 2 )
    ENDIF ELSE BEGIN
        multilineValue = multilineValue + " " + STRTRIM( str, 2 )
    ENDELSE

    IF( STRPOS( multilineValue, "{" ) NE -1 ) AND ( STRPOS( multilineValue, "}" ) NE -1 ) THEN BEGIN
        multilineComplete = 1
    ENDIF

    IF( equalPosition GT 0 ) THEN BEGIN
        name = STRTRIM( STRMID( str, 0, equalPosition ), 2 )
        value = STRTRIM( STRMID( str, equalPosition+1 ), 2 )
        multilineValue = STRTRIM( value, 2 )
    ENDIF ELSE BEGIN
        multilineValue = multilineValue + " " + STRTRIM( str, 2 )
    ENDELSE
multilineValue = STRTRIM(multilineValue, 2)
multilineValue = STRMID(multilineValue, 1, STRLEN(multilineValue)-2)
ENDIF ELSE BEGIN
multilineComplete = 0
ENDELSE

; Parse the name/value pair
CASE STRLOWCASE(name) OF
  "description": IF multilineComplete THEN header.description = multilineValue
  "samples": header.samples = LONG(value)
  "lines": header.lines = LONG(value)
  "bands": header.bands = LONG(value)
  "header offset": header.header_offset = LONG(value)
  "file type": header.file_type = value
  "data type": header.data_type = LONG(value)
  "interleave": header.interleave = STRLOWCASE(value)
  "sensor type": header.sensor_type = value
  "byte order": header.byte_order = LONG(value)
  "wavelength units": header.wavelength_units = value
  "z plot range": IF multilineComplete THEN header.z_plot_range = DOUBLE(STRSPLIT(multilineValue, ",", /EXTRACT))
  "z plot titles": IF multilineComplete THEN header.z_plot_titles = STRSPLIT(multilineValue, ",", /EXTRACT)
  "band names": IF multilineComplete THEN header.band_names = PTR_NEW(STRSPLIT(multilineValue, ",", /EXTRACT))
  "wavelength": IF multilineComplete THEN header.wavelength = PTR_NEW(DOUBLE(STRSPLIT(multilineValue, ",", /EXTRACT)))
ELSE;
ENDCASE
ENDWHILE

; Release the current logical unit number
FREE_LUN, lun

; Return the filled header structure to the calling routine
RETURN, header

END

R.9 READ_ENVI_IMAGE

; NAME:
READ_ENVI_IMAGE

; PURPOSE:
This function serves to read an ENVI image/header directly into...
CHAPTER R. WASP Calibration Code

; an IDL program without the need to first open that image in ENVI
; and use the ENVI_* routines to do so.

; :CATEGORY: Image Processing.

; :CALLING SEQUENCE:
; Result = READ_ENVI_IMAGE( filename, HEADER=header )

; :INPUTS:
; filename: The ENVI image filename to be read.

; :KEYWORD PARAMETERS:
; HEADER: A named variable to receive a structure containing the header
; data read in from the accompanying ENVI image header file.

; :RETURN VALUE:
; An array containing the image data. This array will be of the
; correct data type for the provided image and will appear in BIP
; interleaved order (bands, samples, lines). If any error is
; encountered during this process, the scalar -1 will be returned.

; :SIDE EFFECTS:
; NONE

; :MODIFICATION HISTORY:
; Written by: Carl Salvaggio
; July, 2009 Original code
; December, 2009 OS family dependent swap endian added

; :DISCLAIMER:
; This source code is provided "as is" and without warranties as to performance
; or merchantability. The author and/or distributors of this source code may
; have made statements about this source code. Any such statements do not
; constitute warranties and shall not be relied on by the user in deciding
; whether to use this source code.

; This source code is provided without any express or implied warranties
; whatsoever. Because of the diversity of conditions and hardware under which
; this source code may be used, no warranty of fitness for a particular purpose
; is offered. The user is advised to test the source code thoroughly before
; relying on it. The user must assume the entire risk of using the source code.

FUNCTION READ_ENVI_IMAGE, filename, HEADER=header

; Check for existence of the provided ENVI image file

; IF NOT FILE_TEST( filename ) THEN BEGIN
; MESSAGE, "Image filename provided does not exist", /CONTINUE
; RETURN, -1
; ENDIF

;
R.9. READ_ENVI_IMAGE

; Check for the existence of the default ENVI header file (the ENVI image filename plus the ".hdr" extension), if this does not exist, then provide the user with a dialog pickfile box with which they can locate the appropriate ENVI header file

headerFilename = FILE_DIRNAME(filename, /MARK_DIRECTORY)+FILE_BASENAME(filename, ".img") + ".hdr"

IF NOT FILE_TEST( headerFilename ) THEN BEGIN

headerFilename = DIALOG_PICKFILE( TITLE="Please select ENVI header file", $ PATH=FILE_DIRNAME( filename ), $ FILTER="*.hdr" )

IF ( headerFilename EQ "" ) THEN BEGIN

MESSAGE, "A valid ENVI header file was not provided", /CONTINUE

RETURN, -1

ENDIF

ENDIF

; Define the header structure by reading the ENVI header file

header = READ_ENVI_HEADER( headerFilename )

; Read the image data into a vector of the proper size and data type (skip any header bytes in the image file if they are present)

originalImage = READ_BINARY( filename, $
DATA_DIMS=(header.bands * header.samples * header.
lines), $
DATA_TYPE=header.data_type, $
DATA_START=header.header_offset )

; Reform the vector into an image array according to the interleave type given in the header

CASE header.interleave OF
"bsq": originalImage = REFORM( originalImage, header.samples, header.lines, header.bands )
"bil": originalImage = REFORM( originalImage, header.samples, header.bands, header.lines )
"bip": originalImage = REFORM( originalImage, header.bands, header.samples, header.lines )
ENDCASE

; Rearrange the pixels to BIP format (bands, samples, lines)

IF ( header.interleave NE "bip" ) THEN BEGIN

image = MAKE_ARRAY( header.bands, header.samples, header.lines, TYPE=header. data_type )

FOR band = 0, header.bands-1 DO BEGIN

FOR sample = 0, header.samples-1 DO BEGIN

FOR line = 0, header.lines-1 DO BEGIN

CASE header.interleave OF
"bsq": image[band,sample,line] = originalImage[sample,band,line]
"bil": image[band,sample,line] = originalImage[sample,band,line]
ENDCASE

ENDFOR

ENDFOR

ENDIF
.chapter r. wasp calibration code

115  endcase
116 endfor
117 endfor
118 endfor
119 endif else begin
120  image = originalImage
121  endelse
122
123 ;+ 124 ; Eliminate any unary dimension in the image array
125 ;-
126  image = reform( image )
127
128 ;+ 129 ; Change the endian if necessary
130 ;-
131 ;  case header.byte_order of
132 ;    0: if ( !version.os_family eq "unix" ) then swap_endian_inplace, image
133 ;    1: if ( !version.os_family eq "windows" ) then swap_endian_inplace, image
134 ;    else:
135 ;      endcase
136
137 ;+ 138 ; Return the image to the calling routine
139 ;-
140  return, image
141
142 end
Appendix S

PSO-ALGE Code

S.1 PSO_CLUSTER

```idl
FUNCTION PSO_CLUSTER, generation, particle, config_file, dataPath

  COMPILER_S opt idl2, logical_predicate

  CATCH, err
  IF err NE 0 THEN BEGIN
    CATCH, /CANCEL
    IF N_ELEMENTS(lun) NE 0 THEN FREE_LUN, lun
    MESSAGE, /REISSUE
  ENDIF

  ; Because this function runs on a cluster, you have to restrict IDL to a single thread.
  ; All processing delegation is handled by the job scheduler
  CPU, TPOOL_NTHREADS=1

  PRINT, 'Generation#:' + STRING(generation)
  PRINT, 'Particle#:' + STRING(particle)
  generation = FIX(FLOAT(generation))
  particle = FIX(FLOAT(particle))

  ; Set up all directories
  generation_dir = dataPath + 'generation' + print_generation + '/'
  particle_dir = generation_dir + 'particle' + print_particle + '/'
  generation_prev_dir = dataPath + 'generation' + print_gen_prev + '/'
  analytics_path = dataPath + 'analytics/'
  vinfo_path = analytics_path + 'vinfo/
  pbest_path = analytics_path + 'pbest/
  swarm_path = analytics_path + 'swarm/'
```
gen_dir_exist = FILE_TEST(generation_dir, /DIRECTORY)
IF gen_dir_exist NE 1 THEN FILE_MKDIR, generation_dir

particle_dir_exist = FILE_TEST(particle_dir, /DIRECTORY)
IF particle_dir_exist NE 1 THEN FILE_MKDIR, particle_dir

ana_dir_exist = FILE_TEST(analytics_path, /DIRECTORY)
IF ana_dir_exist NE 1 THEN FILE_MKDIR, analytics_path

vinfo_dir_exist = FILE_TEST(vinfo_path, /DIRECTORY)
IF vinfo_dir_exist NE 1 THEN FILE_MKDIR, vinfo_path

pbest_dir_exist = FILE_TEST(pbest_path, /DIRECTORY)
IF pbest_dir_exist NE 1 THEN FILE_MKDIR, pbest_path

swarm_dir_exist = FILE_TEST(swarm_path, /DIRECTORY)
IF swarm_dir_exist NE 1 THEN FILE_MKDIR, swarm_path

; Read in the configuration file and pull out all the parameters
;PRINT, config_file
params = EXTRACT_PARAMETERS(config_file, particle_dir)
;PRINT, config_file

num_particles = FLOAT(params.NUM_STEPS)
num_parameters = FLOAT(params.NUM_PARAMETERS)
um_generations = FLOAT(params.TOTAL_GENERATIONS)
alge_path = params.ALGE_CONSTANT_PATH
dataPath = params.DATAPATH
ub = FLOAT(params.UB)
lb = FLOAT(params.LB)
initial_fwhm = FLOAT(params.INITIAL_FWHM)
initial_mean = FLOAT(params.INITIAL_MEAN)
opt_mode = FIX(params.OP_MODE)
ratio_flag = FIX(params.RATIO_FLAG)
metric_flag = FIX(params.METRIC_FLAG)
season = FIX(params.SEASON)
w_start = FLOAT(params.W_START)
w_end = FLOAT(params.W_END)
w_varyfor = FLOAT(params.W_VARYFOR)
error_goal = FLOAT(params.ERROR_GOAL)
job_name = params.JOB_NAME
gamma1 = FLOAT(params.GAMMA1)
gamma2 = FLOAT(params.GAMMA2)
minflag = FIX(params.MINFLAG)

ub = FLOAT(ub)
lb = FLOAT(lb)
vmax = 0.15*(ub-lb)/2.0

; Check to see if success file exists. If file exists then success conditions
; have been met and code is exited
exist = FILE_TEST(dataPath+'success.dat')
IF exist EQ 1 THEN BEGIN
PRINT, 'SUCCESS'
RETURN, 200
END
; If this is the first generation and the first particle then initialize everything
IF (generation EQ 0) AND (particle EQ 0) THEN BEGIN
    start_fn = dataPath + 'start.dat'
    OPEN, unit, start_fn, /GET_LUN
    PRINT, unit, SYSTIME(1)
    PRINT, unit, SYSTIME()
    FLUSH, unit
    FREE_LUN, unit
    gbest_fn = dataPath + 'gbest.dat'
    OPEN, unit, gbest_fn, /GET_LUN
    PRINT, unit, 'Beginning'
    FLUSH, unit
    FREE_LUN, unit
    pbest_global_fn = dataPath + 'pbest_global.dat'
    OPEN, unit, pbest_global_fn, /GET_LUN
    FLUSH, unit
    FREE_LUN, unit
    options_fn = dataPath + 'options.dat'
    OPEN, unit, options_fn, /GET_LUN
    PRINT, unit, 'Job name: ' + STRCOMPRESS(STRING(job_name), /REMOVE_ALL)
    PRINT, unit, 'Number of particles: ' + STRCOMPRESS(STRING(num_particles), /REMOVE_ALL)
    PRINT, unit, 'Number of parameters: ' + STRCOMPRESS(STRING(num_parameters)
    FLUSH, unit
    FREE_LUN, unit
CD, generation_dir
FILE_MKDIR, particle_dir

ENDIF

; Determine if particle belongs to the first generation. If particle is from first generation,
; then need to initialize particle with boundaries defined by bounds_arr
IF generation EQ 0 THEN BEGIN
  ; Initialize a velocity if in initial generation and write array to 'vstep.dat'
  ; Add test to see if exists, if doesn't write, if not don't write
  vstep_exist = FILE_TEST(dataPath + 'vstep_000.dat')
  IF particle EQ 0 THEN BEGIN
    IF vstep_exist EQ 1 THEN BEGIN
      PRINT, "generation" + STRING(generation)
      PRINT, "particle" + STRING(particle)
      PRINT, "Vstep exists already and I'm particle 0"
      RETURN, 500
    ENDIF
    PRINT, "First particle"
  ENDIF ELSE BEGIN
    ENDELSE
  Vstep = RANDOMU(seed, num_particles, num_parameters)*vmax
  vstep_fn = dataPath + 'vstep_000.dat'
  OPENW, unit, vstep_fn, /GET_LUN
  PRINTF, unit, Vstep
  FLUSH, unit
  FREE_LUN, unit
  ENDIF ELSE BEGIN

; Define initialized particle based on mean and FWHM
particle_arr = FLTARR(num_parameters)
;PRINT, num_parameters
;HELP, particle_arr
PRINT, 'Calculating initial swarm'
FOR i=0, num_parameters -1 DO BEGIN
  a = RANDOMU(seed)
  particle_arr[i] = a;+1.0
ENDFOR
IF minflag NE 2 THEN BEGIN
  initial_sigma = FLOAT(initial_fwhm)/2.35
  particle_arr = particle_arr * initial_sigma + FLOAT(initial_mean)
ENDIF ELSE BEGIN
  particle_arr = particle_arr*(ub-lb)-(0.5*(ub-lb))
ENDELSE
PRINT, 'Initial particle'
PRINT, particle_arr

PRINT, 'Start running alge/evaluating...
; Send particle to evaluation module for return score
CASE op_mode OF
  0: particle_score = ALGE_EVAL_FLOW(particle_arr, alge_path, particle_dir, ratio_flag, metric_flag, season, seed)
  1: particle_score = ALGE_EVAL_WEATHER(particle_arr, alge_path, particle_dir, ratio_flag, metric_flag, season, seed)
2: particle_score = ALGE_EVAL_ALL(particle_arr, alge_path, particle_dir, ratio_flag, metric_flag, season, seed)

3: particle_score = POSITION2D(particle_arr, minflag)

4: particle_score = ALGE_EVAL_FLOW_HOURLY(particle_arr, alge_path, particle_dir, ratio_flag, metric_flag, season, seed)

5: particle_score = ALGE_EVAL_FLOW_2WEEK(particle_arr, alge_path, particle_dir, ratio_flag, metric_flag, season, seed)

ENDCASE

; Create particle file using naming scheme 'particle-gen#-particle#.dat'
; Write array of data to file [particle, particle score, PBest]
array_towrite = FLTARR(1, num_parameters*2+1)
array_towrite[0, 0:num_parameters-1] = particle_arr
array_towrite[0, num_parameters] = particle_score
array_towrite[0, num_parameters+1:num_parameters*2] = particle_arr
particle_fn = particle_dir + 'generation-' + print_generation + 'particle-' + print_particle + '.dat'

OPENW, unit, particle_fn, /GET_LUN
PRINTF, unit, array_towrite
FLUSH, unit
FREE_LUN, unit

PRINT, 'First generation initialized'
RETURN, 400

ENDIF

; Find all particles from previous generation and store in string array of file names
particle_files = FILE_SEARCH(generation_prev_dir, 'generation-'+ print_gen_prev + 'particle-*')

; Check to make sure the correct number of particles were returned, if not throw error and exit
IF N_ELEMENTS(particle_files) NE num_particles THEN BEGIN
PRINT, "The correct number of particles was not returned"
RETURN, 100
ENDIF

; Create array to hold all data from entire generation of particles
; Each column corresponds to a particle and contains the previous particle parameters, the previous score, and
; the particle's personal best parameter set
particles_data_total = FLTARR(num_particles, num_parameters*2+1)
particle_data = FLTARR(1, num_parameters*2+1)
FOR cur_particle=0, num_particles-1 DO BEGIN
OPENR, unit, particle_files[cur_particle], /GET_LUN
READF, unit, particle_data
particles_data_total[cur_particle, *] = particle_data
FLUSH, unit
FREE_LUN, unit
IF cur_particle EQ 4 THEN data_coming_in = particle_data
ENDFOR
; Extract current generation particles and store in swarm array
swarm = particles_data_total[*, 0:num_parameters-1]
score_array = particles_data_total[*, num_parameters]

; Extract PBest parameter values and store in array
PBest_arr = particles_data_total[*, num_parameters+1:num_parameters*2]
num_parameters_str = STRING(FIX(num_parameters))

IF particle EQ 0 THEN BEGIN
  score_fn = swarm_path + 'score_gen_' + print_gen_prev + '.dat'
  swarm_fn = swarm_path + 'swarm_gen_' + print_gen_prev + '.dat'
  pbest_fn = pbest_path + 'pbest_gen_' + print_gen_prev + '.dat'
  num_parameters_str = STRING(FIX(num_parameters))

  OPENW, unit, swarm_fn, /GET_LUN
  PRINTF, unit, 'Each column represents a parameter, while each row is a particle'
  PRINTF, unit, swarm, FORMAT='('+ num_parameters_str +'(f10.5,5x))'
  FLUSH, unit
  FREE_LUN, unit

  OPENW, unit, score_fn, /GET_LUN
  PRINTF, unit, 'Each row is a particle'
  PRINTF, unit, score_array
  FLUSH, unit
  FREE_LUN, unit

  OPENW, unit, pbest_fn, /GET_LUN
  PRINTF, unit, 'Each column represents a parameter, while each row is a particle'
  PRINTF, unit, PBest_arr, FORMAT='('+ num_parameters_str +'(f10.5,5x))'
  FLUSH, unit
  FREE_LUN, unit

  IF minflag EQ 1 THEN BEGIN
    zero_loc = WHERE((score_array - MIN(score_array)) LE error_goal, zero_count)
  ENDIF ELSE BEGIN
    zero_loc = WHERE((score_array - (0.0)) LE error_goal, zero_count)
  ENDELSE

  IF zero_count GE 1 THEN BEGIN
    zero_exist = FILE_TEST(dataPath + 'converge.dat')
    num_zeros = STRCOMPRESS(STRING(N_ELEMENTS(zero_loc)),/REMOVE_ALL)
    IF zero_exist EQ 1 THEN BEGIN
      count_fn = dataPath + 'count.dat'
      OPENR, unit, count_fn, /GET_LUN
      READF, unit, cur_count
      FLUSH, unit
      FREE_LUN, unit

      zero_fn = dataPath + 'converge.dat'
      OPENU, unit, zero_fn, /GET_LUN, /APPEND
      PRINTF, unit, STRCOMPRESS(STRING(zero_count), /REMOVE_ALL) + ' particle(s) in generation: ' + STRCOMPRESS(STRING(generation), /REMOVE_ALL) + ' have achieved minimum.'
      PRINTF, unit, 'Particle ID(s) that have achieved minimum: '
  ENDIF ELSE BEGIN
PRINTF, unit, TRANSPOSE(zero_loc), FORMAT="('+ num_zeros +'(i3,5x ))'
PRINTF, unit, 'Current minimum score(s):'
PRINTF, unit, TRANSPOSE(score_array[zero_loc]), FORMAT="('+ num_parameters_str +'(f10.5,5x))'
PRINTF, unit, 'Current personal best of particle(s) at minimum:
PRINTF, unit, TRANSPOSE(Pbest_arr[zero_loc, *]), FORMAT="('+ num_parameters_str +'(f10.5,5x))'
FLUSH, unit
FREE_LUN, unit

count_fn = dataPath + 'count.dat'
OPENW, unit, count_fn, /GET_LUN, /APPEND
PRINTF, unit, zero_count
FLUSH, unit
FREE_LUN, unit
ENDIF ELSE BEGIN

OPENW, unit, count_fn, /GET_LUN, /APPEND
PRINTF, unit, zero_count
FLUSH, unit
FREE_LUN, unit

zero_fn = dataPath + 'converge.dat'
OPENW, unit, zero_fn, /GET_LUN
PRINTF, unit, 'Each column represents a parameter, while each row
is a particle'
PRINTF, unit, STRCOMPRESS(STRING(zero_count), /REMOVE_ALL) +'
  particle(s) in generation: '+$
  print_generation + ' have achieved minimum'
PRINTF, unit, zero_loc, FORMAT="('+ num_zeros +'(i3,5x))'
PRINTF, unit, 'Current minimum score(s):'
PRINTF, unit, TRANSPOSE(score_array[zero_loc]), FORMAT="('+ num_parameters_str +'(f10.5,5x))'
PRINTF, unit, 'Current personal best of particle(s) at minimum:
PRINTF, unit, TRANSPOSE(Pbest_arr[zero_loc, *]), FORMAT="('+ num_parameters_str +'(f10.5,5x))'
FLUSH, unit
FREE_LUN, unit
ENDELSE
; If solution exists which satisfies overall error goal, then
optimization is complete. Write success file and exit
IF zero_count EQ num_particles THEN BEGIN
end_time = SYSTIME(1)
end_time_print = SYSTIME()
start_fn = dataPath + 'start.dat'
start_time = STRARR(2)
OPENR, unit, start_fn, /GET_LUN
READF, unit, start_time
FLUSH, unit
FREE_LUN, unit
start_time_print = start_time[1]
OPENW, unit, dataPath+'success.dat', /GET_LUN
PRINTF, unit, 'Convergence by all particles in generation: '+
   print_gen_prev
PRINT, unit, PBest_arr
PRINT, unit, 'Proces begins at: ' + start_time_print
PRINT, unit, 'Process converged at: ' + end_time_print
FLUSH, unit
FREE_LUN, unit
RETURN, 600
ENDIF
ENDIF
ENDIF
ENDIF
; Determine where Global best particle is located based on the score stored in
; the particle data arra
IF minflag EQ 1 THEN BEGIN
    Best_particle_loc = WHERE(score_array EQ MIN(score_array))
ENDIF ELSE BEGIN
    Best_particle_loc = WHERE(score_array EQ MIN(score_array))
ENDELSE
Best_particle_idx = ARRAY_INDICES(score_array, Best_particle_loc)
; Extract both the particle parameters and the score for the Global best
; particle
Best_particle = PBest_arr[Best_particle_idx[0], *]
PRINT, 'BEST PARTICLE'
PRINT, Best_particle
GBest_arr = REPLICATE(1, num_particles) # Best_particle
OPENU, unit, dataPath+'gbest.dat', /GET_LUN, /APPEND
PRINTF, unit, 'Particle#: ' + STRCOMPRESS(STRING(Best_particle_loc), /REMOVE_ALL) + ' in generation#: ' + print_generation + ' with a score of: ' + STRCOMPRESS(STRING(score_array[best_particle_idx[0]]), /REMOVE_ALL)
FREE_LUN, unit
; Determine the value of weight change
w_increment = (w_start - w_end)/num_generations
w_now = w_start - (w_increment*generation)
; Generate random weighted stochastic variables
alpha1 = RANDOMU(seed)
alpha2 = RANDOMU(seed)
; Read in previous generation VSTEP from vstep.dat file
; Make file vstep-generation and then pull in individual vstep file per particle
vstep_current_fn = 'vstep_' + print_generation
vstep_previous_fn = 'vstep_' + print_gen_prev
; If I am particle 0 then I am responsible for creating the current vstep file
IF particle EQ 0 THEN BEGIN
    ; Otherwise, create the current vstep file by reading in the previous vstep
    ; information
    Vstep = FLTARR(num_particles, num_parameters)
    vstep_fn = dataPath + 'vstep_' + print_gen_prev + '.dat'
    OPENR, unit, vstep_fn, /GET_LUN
    READF, unit, Vstep
**S.1. PSO_CLUSTER**

```plaintext
; Calculate velocity
k=1.0
\gamma_1 = 2.05
\gamma_2 = 2.05
\var = \gamma_1 + \gamma_2
\text{constiction\_coeff} = \frac{(2.0 \times k)}{\text{ABS}(\var - 2.0 - \sqrt{\var \times (\var - 4.0)})}

\text{IF} \minflag \text{EQ} 2 \text{ THEN BEGIN} \\
\ w_{\text{now}} = 0.729844 \\
\ \gamma_1 = 1.496180 \\
\ \gamma_2 = 1.496180 \\
\text{ENDIF}

\text{Vstep} = w_{\text{now}} \times \text{Vstep} + \gamma_1 \times \alpha_1 \times (\text{PBest\_arr} - \text{swarm}) + \gamma_2 \times \alpha_2 \times (\text{GBest\_arr} - \text{swarm}) \\
\text{Vstep} = \text{constiction\_coeff} \times (\text{Vstep} + \gamma_1 \times \alpha_1 \times (\text{PBest\_arr} - \text{swarm}) + \gamma_2 \times \alpha_2 \times (\text{GBest\_arr} - \text{swarm}))

; Apply Vmax operator for v > Vmax \\
\text{changeRows} = \text{WHERE}(\text{Vstep} \text{ GT} \ \text{vmax}) \\
\text{IF} \text{changeRows}[0] \text{ NE} -1 \text{ THEN Vstep}[\text{changeRows}] = \text{vmax} \\
\text{IF} \text{changeRows}[0] \text{ NE} -1 \text{ THEN Vstep}[\text{changeRows}] = 0.0

; Apply Vmax operator for v < -Vmax \\
\text{changeRows} = \text{WHERE}(\text{Vstep} \text{ LT} \ -\text{vmax}) \\
\text{IF} \text{changeRows}[0] \text{ NE} -1 \text{ THEN Vstep}[\text{changeRows}] = -\text{vmax} \\
\text{IF} \text{changeRows}[0] \text{ NE} -1 \text{ THEN Vstep}[\text{changeRows}] = 0.0

\text{vstep\_fn} = \text{dataPath} + '\text{vstep}_' + \text{print\_generation} + '.dat'
\text{OPENW, unit, vstep\_fn, /GET\_LUN}
\text{PRINTF, unit, Vstep}
\text{FLUSH, unit}
\text{FREE\_LUN, unit}

; If I am not particle 0, then the vstep file should already exist
\text{ENDIF ELSE BEGIN} \\
\text{vstep\_current\_fn} = '\text{vstep}_' + \text{print\_generation}
\text{vstep\_exist} = \text{FILE\_TEST(dataPath + vstep\_current\_fn + '.dat')}
\text{WHILE vstep\_exist NE 1 DO BEGIN} \\
\ \text{WAIT}, 5 \\
\ \text{vstep\_exist} = \text{FILE\_TEST(dataPath + vstep\_current\_fn + '.dat')}
\text{PRINT, 'Waiting vstep to be created'}
\text{ENDWHILE}

\text{PRINT, vstep\_current\_fn}
\text{PRINT, vstep\_previous\_fn}

\text{Vstep} = \text{FLTARR(num\_particles, num\_parameters)}
\text{vstep\_fn} = \text{dataPath} + '\text{vstep}_' + \text{print\_generation} + '.dat'
\text{OPENR, unit, vstep\_fn, /GET\_LUN}
\text{READF, unit, Vstep}
\text{FLUSH, unit}
\text{FREE\_LUN, unit}
\text{ENDELS}
```
;PRINT, vstep_current_fn
;PRINT, vstep_previous_fn

; Update particle positions
swarm = swarm + Vstep

; Check to see if swarm has gone outside of boundaries, if so, change those
particles to in bound
outofbounds_high = WHERE(swarm GE ub)
outofbounds_low = WHERE(swarm LE lb)

IF outofbounds_high[0] NE -1 THEN swarm[outofbounds_high] = ub
IF outofbounds_low[0] NE -1 THEN swarm[outofbounds_low] = lb

; Send particle to evaluation module for return score
eval_particle = swarm[particle, *]
CASE op_mode OF

0: particle_score = ALGE_EVAL_FLOW(eval_particle, alge_path, particle_dir,
   ratio_flag, metric_flag, season, seed)
1: particle_score = ALGE_EVAL_WEATHER(eval_particle, alge_path,
   particle_dir, ratio_flag, metric_flag, season, seed)
2: particle_score = ALGE_EVAL_ALL(eval_particle, alge_path, particle_dir,
   ratio_flag, metric_flag, season, seed)
3: particle_score = POSITION2d(eval_particle, minflag)
4: particle_score = ALGE_EVAL_FLOW_HOURLY(eval_particle, alge_path,
   particle_dir, ratio_flag, metric_flag, season, seed)
5: particle_score = ALGE_EVAL_FLOW_2WEEK(eval_particle, alge_path,
   particle_dir, ratio_flag, metric_flag, season, seed)
ENDCASE

previous_score = score_array[particle]

IF particle EQ 0 THEN BEGIN
OPENU, unit, dataPath + 'pbest_global.dat', /APPEND, /GET_LUN
PRINTF, unit, previous_score
FLUSH, unit
FREE_LUN, unit
ENDIF

; Determine if new particle score is better than personal best score. Write
; particle parameters, returned score,
; and particle's personal best parameter values to next generation particle file
.
IF previous_score LT particle_score THEN BEGIN
PBest_towrite = PBest_arr[particle, *]
score_towrite = previous_score
ENDIF ELSE BEGIN
PBest_towrite = eval_particle
score_towrite = particle_score
ENDElse
array_towrite = FLTARR(1, num_parameters*2+1)
array_towrite[0, 0:num_parameters-1] = eval_particle
array_towrite[0, num_parameters] = score_towrite
array_towrite[0, num_parameters+1:num_parameters*2] = PBest_towrite
particle_fn = particle_dir + ‘generation-‘ + print_generation + ‘particle-‘ + $
               print_particle + ‘.dat’
OPENW, unit, particle_fn, /GET_LUN
PRINTF, unit, array_towrite
FLUSH, unit
FREE_LUN, unit
IF (generation EQ num_generations-1) AND (exist NE 1) AND (particle EQ
num_particles-1) THEN BEGIN
  end_time = SYSTIME(1)
  end_time_print = SYSTIME()
  start_fn = dataPath + ‘start.dat’
  start_time = STRARR(2)
  OPENR, unit, start_fn, /GET_LUN
  READF, unit, start_time
  FLUSH, unit
  FREE_LUN, unit
  start_time_print = start_time[1]
  OPENW, unit, dataPath+’no_success.dat’, /GET_LUN
  PRINTF, unit, ‘Convergence not achieved by ‘ + STRCOMPRESS(STRING(
                       generation+1), /REMOVE_ALL) + ‘ generations.’
  PRINTF, unit, ‘Best swarm achieved:’
  PRINTF, unit, PBest_arr
  PRINTF, unit, ‘With a score array of :’
  PRINTF, unit, score_array
  PRINTF, unit, ‘Process began at: ‘ + start_time_print
  PRINTF, unit, ‘Process converged at: ‘ + end_time_print
  FLUSH, unit
  FREE_LUN, unit
  RETURN, 900
ENDIF
END

S.2 ALGE_EVAL_ALL

FUNCTION ALGE_EVAL_ALL, particle, alge_path, new_data_path, ratio_flag, metric_flag,
season, seed

  ratio_array = [RANDOMU(seed), RANDOMU(seed)]
  CASE ratio_flag OF
    1: RETURN, ratio_array[0]
    2: RETURN, ratio_array[1]
    3: RETURN, 0.5*ratio_array[0]+0.5*ratio_array[1]
  ENDCASE
CD, new_data_path
FILE_COPY, [‘algesal’, ‘*.dat’, ‘dimar.inc’], alge_path
weatherFileName = new_data_path + 'sfc.dat'
flowFileName = new_data_path + 'flow.dat'
tempFileName = new_data_path + 'deltat.dat'
windSpd_delta = particle[0]
windDir_delta = particle[1]
cloudF_delta = particle[2]
cloudH_delta = particle[3]
airT_delta = particle[4]
dewPt_delta = particle[5]
snowAlbido_delta = particle[6]
flow_delta = particle[7]
temp_delta = particle[8]
pressure_delta = particle[9]

; Open meteorology file and read in weather data to string array
nlines = FILE_LINES(weatherFileName)
weather_data_str = STRARR(nlines)
OPENR, unit, weatherFileName, /GET_LUN
READF, unit, weather_data_str
FREE_LUN, unit

; Open flow file and read in flow data to string array
nlinesF = FILE_LINES(flowFileName)
flow_data_str = STRARR(nlinesF)
OPENR, unit, flowFileName, /GET_LUN
READF, unit, flow_data_str
FREE_LUN, unit

; Open temp file and read in temp data to string array
nlinesT = FILE_LINES(tempFileName)
temp_data_str = STRARR(nlinesT)
OPENR, unit, tempFileName, /GET_LUN
READF, unit, temp_data_str
FREE_LUN, unit

; Create float arrays to hold weather data columns
hours_data = INTARR(nlines-1)
windDir_data = INTARR(nlines-1)
windSpd_data = FLTARR(nlines-1)
airT_data = INTARR(nlines-1)
dewPt_data = INTARR(nlines-1)
cloudF_data = FLTARR(nlines-1)
cloudH_data = FLTARR(nlines-1)
pressure_data = FLTARR(nlines-1)
snowAlbido_data = FLTARR(nlines-1)
date_data = STRARR(nlines-1)
time_data = STRARR(nlines-1)

; Loop through string array of weather data and read columns of data into allotted arrays
weather_data_str = STRCOMPRESS(weather_data_str)
FOR curLine=0, nlines-2 DO BEGIN
    extracted = STRSPLIT(weather_data_str[curLine+1], ' ', /EXTRACT)
    hours_data[curLine] = FIX(extracted[0])
    windDir_data[curLine] = FIX(extracted[1])
    windSpd_data[curLine] = FLOAT(extracted[2])
airT_data[curline] = FIX(extracted[3])
dewPt_data[curline] = FIX(extracted[4])
cloudF_data[curline] = FLOAT(extracted[5])
cloudH_data[curline] = FLOAT(extracted[6])
pressure_data[curline] = FLOAT(extracted[7])
snowAlbido_data[curline] = FLOAT(extracted[8])
date_data[curline] = extracted[9]
time_data[curline] = extracted[10]
ENDFOR

windDir_data_new = windDir_data + windDir_data * windDir_delta
windSpd_data_new = windSpd_data + windSpd_data * windSpd_delta
airT_data_new = airT_data + airT_data * airT_delta
dewPt_data_new = dewPt_data + dewPt_data * dewPt_delta
cloudF_data_new = cloudF_data + cloudF_data * cloudF_delta
cloudH_data_new = cloudH_data + cloudH_data * cloudH_delta
pressure_data_new = pressure_data + pressure_data * pressure_delta
snowAlbido_data_new = snowAlbido_data + snowAlbido_data * snowAlbido_delta

flow_data = FLOAT(flow_data_str)
flow_data_new = flow_data + flow_data * flow_delta
temp_data = FLOAT(temp_data_str)
temp_data_new = temp_data + temp_data * temp_delta

; Apply wind direction constraint
; Wind direction cannot be negative
negative_wd = WHERE(windDir_data_new LT 0.0, count)
IF count NE 0 THEN windDir_data_new[negative_wd] = 0.0

; Apply wind speed constraint
; Wind speed cannot be negative
negative_ws = WHERE(windSpd_data_new LT 0.0, count)
IF count NE 0 THEN windSpd_data_new[negative_ws] = 0.0

; Apply dew point constraint
; Dew point cannot be higher than air temperature
high_idx_dp = WHERE(dewPt_data_new GT airT_data_new, count)
IF count NE 0 THEN dewPt_data_new[high_idx_dp] = airT_data_new[high_idx_dp]

; Apply cloud fraction constraint
; Cloud fraction must be > than 0 and < 1.0
outofboundsHI_cf = WHERE(cloudF_data_new GT 1.0, count)
IF count NE 0 THEN cloudF_data_new[outofboundsHI_cf] = 1.0
outofboundsLO_cf = WHERE(cloudF_data_new LT 0.0, count)
IF count NE 0 THEN cloudF_data_new[outofboundsLO_cf] = 0.0

; Apply snow albido constraint
; Snow albido must be > than 0 and < 1.0
outofboundsHI_sa = WHERE(snowAlbido_data_new GT 1.0, count)
IF count NE 0 THEN snowAlbido_data_new[outofboundsHI_sa] = 1.0
outofboundsLO_sa = WHERE(snowAlbido_data_new LT 0.0, count)
IF count NE 0 THEN snowAlbido_data_new[outofboundsLO_sa] = 0.0

; Apply flow constraint
; Flow cannot be negative
negative_flow = WHERE(flow_data_new LT 0.0, count)
IF count NE 0 THEN flow_data_new[negative_flow] = 0.0

; Apply temperature difference constraint
; Temperature difference across input/output cannot be negative
IF count NE 0 THEN temp_data_new[negative_temp] = 0.0

sfc_newfilename = new_data_path + 'sfc.dat'
flow_newfilename = new_data_path + 'flow.dat'
temp_newfilename = new_data_path + 'deltat.dat'

OPENW, unit, sfc_newfilename, /GET_LUN
PRINTF, unit, weather_data_str[0]
FOR curLine=0L, nlines-2 DO BEGIN
  str = STRING(STRCOMPRESS(hours_data[curline], /REMOVE_ALL)) + ' ' +
  STRING(STRCOMPRESS(windDir_data_new[curline], /REMOVE_ALL)) +
  ' ' +
  STRING(STRCOMPRESS(windSpd_data_new[curline], /REMOVE_ALL)) +
  ' ' +
  STRING(STRCOMPRESS(airT_data_new[curline], /REMOVE_ALL)) +
  ' ' +
  STRING(STRCOMPRESS(dewPt_data_new[curline], /REMOVE_ALL)) +
  ' ' +
  STRING(STRCOMPRESS(cloudF_data_new[curline], /REMOVE_ALL)) +
  ' ' +
  STRING(STRCOMPRESS(cloudH_data_new[curline], /REMOVE_ALL)) +
  ' ' +
  STRING(STRCOMPRESS(pressure_data_new[curline], /REMOVE_ALL)) +
  ' ' +
  STRING(STRCOMPRESS(snowAlbido_data_new[curline], /REMOVE_ALL)) +
  ' ' +
  STRING(STRCOMPRESS(time_data[curline], /REMOVE_ALL))
  PRINTF, unit, str, FORMAT='(a125)'
ENDFOR
FREE_LUN, unit

OPENW, unit, flow_newfilename, /GET_LUN
FOR curLine=0L, nlinesF-1 DO BEGIN
  PRINTF, unit, flow_data_new[curLine]
ENDFOR
FREE_LUN, unit

OPENW, unit, temp_newfilename, /GET_LUN
FOR curLine=0L, nlinesT-1 DO BEGIN
  PRINTF, unit, temp_data_new[curLine]
ENDFOR
FREE_LUN, unit

SPAWN, './algesal'

images_success = RUN_MAKE_ALGE_IMAGE(season, new_data_path)
FOR ratio_flag = 1 TO n_ratio DO BEGIN
  ratio_array = [RANDOMU(seed1), RANDOMU(seed2)]
  PRINTF, unit, temp_data_new[curLine]
ENDFOR
FREE_LUN, unit

CASE ratio_flag OF
S.3 ALGE_EVAL_FLOW

FUNCTION ALGE_EVAL_FLOW, particle, alge_path, new_data_path, ratio_flag, metric_flag, season, seed

; ratio_array = [RANDOMU(seed), RANDOMU(seed)]
CASE ratio_flag OF
  1: RETURN, ratio_array[0]
  2: RETURN, ratio_array[1]
  3: RETURN, 0.5*ratio_array[0]+0.5*ratio_array[1]
ENDCASE

CD, new_data_path
FILE_COPY, ['algesal', '*.dat', 'dimar.inc'], alge_path
flowFileName = new_data_path + 'flow.dat'

; Open flow file and read in flow data to string array
nlinesF = FILE_LINES(flowFileName)
flow_data_str = STRARR(nlinesF)
OPENR, unit, flowFileName, /GET_LUN
READF, unit, flow_data_str
FREE_LUN, unit

; Each element of the particle array represents the value that needs to be assigned to
; each window segment in the flow file array. These values are randomly generated and
; influenced by the swarm. The bounds of these values are user defined in the
; configuration file "configuration.sh"
num_new_pts = N_ELEMENTS(particle)
num_cur_pts = nlinesF
IF num_cur_pts MOD num_new_pts NE 0 THEN BEGIN
  PRINT, "User hasn't chosen an appropriate window size in the configuration file"
  PRINT, "for the number of entries in the flow file."
  RETURN, 800
ENDIF
window_size = num_cur_pts/num_new_pts
flow_data_new = FLTARR(num_cur_pts)
FOR i=0, num_new_pts-1 DO BEGIN
  data_block = REPLICATE(particle[i], window_size)
  flow_data_new[i*window_size:(i+1)*window_size-1] = MEAN(data_block)
ENDFOR

; Apply flow constraint
; Flow cannot be negative
negative_flow = WHERE(flow_data_new LT 0.0, count)
IF count NE 0 THEN flow_data_new[negative_flow] = 0.0
flow_newfilename = new_data_path + 'flow.dat'
OPENW, unit, flow_newfilename, /GET_LUN
FOR curLine=0L, nlinesF-1 DO BEGIN
  PRINTF, unit, flow_data_new[curLine]
ENDFOR
FREE_LUN, unit
SPAWN, './algesal'
images_success = RUN_MAKE_ALGE_IMAGE(season, new_data_path)
ratio_array = RUN_METRIC_ENGINE(season, alge_path, new_data_path, metric_flag)
PRINT, ratio_array
CASE ratio_flag OF
  1: RETURN, ratio_array[0]
  2: RETURN, ratio_array[1]
  3: RETURN, 0.5*ratio_array[0]+0.5*ratio_array[1]
ENDCASE
PRINT, 'alge path: ' + alge_path
PRINT, 'data path: ' + new_data_path
CD, new_data_path
FILE_COPY, alge_path + 'algesal', new_data_path
FILE_COPY, alge_path + '*.dat', new_data_path
FILE_COPY, alge_path + 'dimar.inc', new_data_path
flowFileName = new_data_path + 'flow.dat'
; Open flow file and read in flow data to string array
nlinesF = FILE_LINES(flowFileName)
flow_data_str = STRARR(nlinesF)
OPENR, unit, flowFileName, /GET_LUN
READF, unit, flow_data_str
FREE_LUN, unit

; Each element of the particle array represents the value that needs to be assigned to
; each window segment in the flow file array. These values are randomly generated and
; influenced by the swarm. The bounds of these values are user defined in the
num_new_pts = N_ELEMENTS(particle)
num_cur_pts = nlinesF
PRINT, 'num elements in particle: ' + STRING(num_new_pts)
PRINT, 'num lines in flow file: ' + STRING(num_cur_pts)
IF num_cur_pts MOD num_new_pts NE 0 THEN BEGIN
  PRINT, "User hasn’t chosen an appropriate window size in the configuration file"
  PRINT, "for the number of entries in the flow file."
  RETURN, 800
ENDIF
window_size = num_cur_pts/num_new_pts
flow_data_new = FLTARR(num_cur_pts)
FOR i=0, num_new_pts-1 DO BEGIN
  data_block = REPLICATE(particle[i], window_size)
  flow_data_new[i*window_size:(i+1)*window_size-1] = data_block
ENDFOR
PRINT, 'num elements in new flow file: ' + STRING(N_ELEMENTS(flow_data_new))

flow_newfilename = new_data_path + 'flow.dat'
OPENW, unit, flow_newfilename, /GET_LUN
FOR curLine=0L, nlinesF-1 DO BEGIN
  PRINTF, unit, flow_data_new[curLine]
ENDFOR
FREE_LUN, unit
SPAWN, './algesal'
images_success = RUN_MAKE_ALGE_IMAGE(season, new_data_path, alge_path, /ICE)
ratio_array = RUN_METRIC_ENGINE(season, alge_path, new_data_path, metric_flag)
ratio_array = READ_ALL_ALGE_DATA_ICE(alge_path, metric_flag, new_data_path)
ratio_ice = METRIC_ICE(alge_path, new_data_path, metric_flag)
ratio_array = [ratio_ice, 0]
PRINT, ratio_array
CASE ratio_flag OF
  1: RETURN, ratio_array[0]
  2: RETURN, ratio_array[1]
  3: RETURN, 0.5*ratio_array[0]+0.5*ratio_array[1]
ENDCASE
FUNCTION ALGE_EVAL_WEATHER, particle, alge_path, new_data_path, ratio_flag, metric_flag, season

CD, new_data_path
FILE_COPY, ['algesal', '*.dat', 'dimar.inc'], alge_path

weatherFileName = new_data_path + 'sfc.dat'
windspeed_delta = particle[0]
winddir_delta = particle[1]
cloudF_delta = particle[2]
cloudH_delta = particle[3]
airT_delta = particle[4]
dewPt_delta = particle[5]
snowAlbido_delta = particle[6]
pressure_delta = particle[7]

; Open meteorology file and read in weather data to string array
nlines = FILE_LINES(weatherFileName)
weather_data_str = STRARR(nlines)
OPENR, unit, weatherFileName, /GET_LUN
READF, unit, weather_data_str
FREE_LUN, unit

; Create float arrays to hold weather data columns
hours_data = INTARR(nlines-1)
windSpd_data = FLTARR(nlines-1)
airT_data = INTARR(nlines-1)
dewPt_data = INTARR(nlines-1)
cloudF_data = FLTARR(nlines-1)
cloudH_data = FLTARR(nlines-1)
pressure_data = FLTARR(nlines-1)
snowAlbido_data = FLTARR(nlines-1)
date_data = STRARR(nlines-1)
time_data = STRARR(nlines-1)

; Loop through string array of weather data and read columns of data into allotted arrays
weather_data_str = STRCOMPRESS(weather_data_str)
FOR curLine=0, nlines-2 DO BEGIN
    extracted = STRSPLIT(weather_data_str[curLine+1], ' ', /EXTRACT)
    hours_data[curLine] = FIX(extracted[0])
    windSpd_data[curLine] = FLOAT(extracted[2])
    airT_data[curLine] = FIX(extracted[3])
    dewPt_data[curLine] = FIX(extracted[4])
    cloudF_data[curLine] = FLOAT(extracted[5])
    cloudH_data[curLine] = FLOAT(extracted[6])
    pressure_data[curLine] = FLOAT(extracted[7])
    snowAlbido_data[curLine] = FLOAT(extracted[8])
    date_data[curLine] = extracted[9]
    time_data[curLine] = extracted[10]
ENDFOR

windSpd_data_new = windSpd_data + windSpd_data * windSpd_delta
airT_data_new = airT_data + airT_data * airT_delta
dewPt_data_new = dewPt_data + dewPt_data * dewPt_delta
cloudF_data_new = cloudF_data + cloudF_data * cloudF_delta
cloudH_data_new = cloudH_data + cloudH_data * cloudH_delta
pressure_data_new = pressure_data + pressure_data * pressure_delta
snowAlbido_data_new = snowAlbido_data + snowAlbido_data * snowAlbido_delta

; Apply wind direction constraint
; Wind direction cannot be negative
negative_wd = WHERE(windDir_data_new LT 0.0, count)
IF count NE 0 THEN windDir_data_new[negative_wd] = 0.0

; Apply wind speed constraint
; Wind speed cannot be negative
negative_ws = WHERE(windSpd_data_new LT 0.0, count)
IF count NE 0 THEN windSpd_data_new[negative_ws] = 0.0

; Apply dew point constraint
; Dew point cannot be higher than air temperature
high_idx_dp = WHERE(dewPt_data_new GT airT_data_new, count)
IF count NE 0 THEN dewPt_data_new[high_idx_dp] = airT_data_new[high_idx_dp]

; Apply cloud fraction constraint
; Cloud fraction must be > than 0 and < 1.0
outofboundsHI_cf = WHERE(cloudF_data_new GT 1.0, count)
outofboundsLO_cf = WHERE(cloudF_data_new LT 0.0, count)
IF count NE 0 THEN cloudF_data_new[outofboundsHI_cf] = 1.0
IF count NE 0 THEN cloudF_data_new[outofboundsLO_cf] = 0.0

; Apply snow albedo constraint
; Snow albedo must be > than 0 and < 1.0
outofboundsHI_sa = WHERE(snowAlbido_data_new GT 1.0, count)
outofboundsLO_sa = WHERE(snowAlbido_data_new LT 0.0, count)
IF count NE 0 THEN snowAlbido_data_new[outofboundsHI_sa] = 1.0
IF count NE 0 THEN snowAlbido_data_new[outofboundsLO_sa] = 0.0

sfc_newfilename = new_data_path + 'sfc.dat'

OPENW, unit, sfc_newfilename, /GET_LUN
PRINTF, unit, weather_data_str[0]
FOR curLine=0L, nlines-2 DO BEGIN
str = STRING(STRCOMPRESS(hours_data[curline], /REMOVE_ALL)) + ' ' + STRING(STRCOMPRESS(windDir_data_new[curline], /REMOVE_ALL)) + ' ' + STRING(STRCOMPRESS(windSpd_data_new[curline], /REMOVE_ALL)) + ' ' + STRING(STRCOMPRESS(airT_data_new[curline], /REMOVE_ALL)) + ' ' + STRING(STRCOMPRESS(dewPt_data_new[curline], /REMOVE_ALL)) + ' ' + STRING(STRCOMPRESS(cloudF_data_new[curline], /REMOVE_ALL)) + ' ' + STRING(STRCOMPRESS(cloudH_data_new[curline], /REMOVE_ALL)) + ' ' + STRING(STRCOMPRESS(pressure_data_new[curline], /REMOVE_ALL)) + ' ' + STRING(STRCOMPRESS(snowAlbido_data_new[curline], /REMOVE_ALL)) + ' ' + STRING(STRCOMPRESS(date_data[curline], /REMOVE_ALL)) + ' ' + STRING(STRCOMPRESS(time_data[curline], /REMOVE_ALL))
PRINTF, unit, str, FORMAT='(a125)'
ENDFOR
FREE_LUN, unit

SPAWN, './algesal'

images_success = RUN_MAKE_ALGE_IMAGE(season, new_data_path)
ratio_array = RUN_METRIC_ENGINE(season, alge_path, new_data_path, metric_flag)
S.7 ALGE_METRIC

FUNCTION ALGE_METRIC, simulated, observed, metric_flag
CASE metric_flag OF
  1: BEGIN
    avg_simulated = MEAN(simulated)
    avg_observed = MEAN(observed)
    nomean_simulated = simulated - avg_simulated
    nomean_observed = observed - avg_observed
    nomean_sim2 = nomean_simulated^2
    nomean_obs2 = nomean_observed^2
    sim_obs = nomean_simulated*nomean_observed
    Ra_top = TOTAL(nomean_sim2-2*sim_obs+nomean_obs2)
    Ra_bot = TOTAL(nomean_sim2+nomean_obs2)
    R_a = SQRT(Ra_top/Ra_bot)
    RETURN, R_a
  2: BEGIN
    diff = simulated - observed
    diff_squared = diff^2
    sum = TOTAL(diff_squared)
    square_root = SQRT(sum/N_ELEMENTS(diff))
    RMS_norm = square_root/(MAX(observed)-MIN(observed))
    RETURN, RMS_norm
  ENDENDCASE
END

S.8 CALC_DELTA_T

FUNCTION CALC_DELTA_T, temp_img, path, hour
sfc_file_fn = FILE_WHICH(path, 'sfc.dat')
nlines_sfc = FILE_LINES(sfc_file_fn)
sfc = STRARR(nlines_sfc)
OPENR, unit, sfc_file_fn, /GET_LUN
READF, unit, sfc
FREE_LUN, unit
water_nodes = WHERE(temp_img GT 273.160004, count)
avgT = TOTAL(temp_img[water_nodes])/count
extracted = STRSPLIT(sfc[hour], /EXTRACT)
airT = FLOAT(extracted[3]) + 273.15

  PRINT, 'a'
ENDIF
RETURN, (avgT - airT)

S.9 CALC_ICE_COVERAGE

FUNCTION CALC_ICE_COVERAGE, ice_image, alge_path
igrid_fn = FILE_WHICH(alge_path, 'igrid.dat')
command_one = 'grep 1 -o ' + igrid_fn + ' | wc -l'
command_zero = 'grep 0 -o ' + igrid_fn + ' | wc -l'
SPAWN, command_one, num_ones
SPAWN, command_zero, num_zeros
num_ice_nodes = WHERE(ice_image GT 0, count)

percent_coverage = FLOAT(count)/FLOAT(num_ones)
RETURN, percent_coverage

S.10 EXTRACT_PARAMETERS

FUNCTION EXTRACT_PARAMETERS, config_file, particle_dir
FILE_COPY, config_file, particle_dir+'config.sh', /OVERWRITE
CD, particle_dir
config = FILE_SEARCH('config.sh')
keys = ['alge_constant_path', 'dataPath', 'JOB_NAME',
        'NUM_STEPS', 'num_parameters',
        'TOTAL_GENERATIONS', 'error_goal',
        'minflag', 'gamma1',
        'gamma2', 'w_start',
        'w_end']
FUNCTION EXTRACT_ICE_IMGS, season, path

IF season EQ 0 THEN BEGIN
    date_arr = [[2, 24, 2009, 12, 0, 0], [2, 16, 2009, 13, 0, 0], [3, 4, 2009, 13, 0, 0]]
ENDIF ELSE BEGIN
    date_arr = [[2, 11, 2010, 12, 0, 0], [2, 11, 2010, 21, 0, 0], [3, 4, 2010, 15, 0, 0], [3, 4, 2010, 21, 0, 0]]
ENDELSE

s = SIZE(date_arr, /DIMENSIONS)
num_days = s[1]
ice_list = LIST()
FOR i=0, num_days-1 DO BEGIN
    date = date_arr[i, *]
    ice_img_fn = path + 'iceimg' + STRCOMPRESS(STRING(date[0]), /REMOVE_ALL) + '_' + STRCOMPRESS(STRING(date[1]), /REMOVE_ALL) + '_' + STRCOMPRESS(STRING(date[2]), /REMOVE_ALL) + '_' + STRCOMPRESS(STRING(date[3]), /REMOVE_ALL) + '.tif'
ENDFOR
S.12 EXTRACT_TEMP_IMGS

FUNCTION EXTRACT_TEMP_IMGS, season, path

IF season EQ 0 THEN BEGIN
    date_arr = [[2, 24, 2009, 12, 0, 0], [2, 16, 2009, 13, 0, 0], [3, 4, 2009, 13, 0, 0]]
ENDIF ELSE BEGIN
    date_arr = [[2, 11, 2010, 12, 0, 0], [2, 11, 2010, 21, 0, 0], [3, 4, 2010, 15, 0, 0], [3, 4, 2010, 21, 0, 0]]
ENDELSE

s = SIZE(date_arr, /DIMENSIONS)
num_days = s[1]
temp_list = LIST()
FOR i=0, num_days-1 DO BEGIN
    date = date_arr[*, i]
    temp_img_fn = path + 'tempimg' + STRCOMPRESS(STRING(date[0]), /REMOVE_ALL) + '_' + STRCOMPRESS(STRING(date[1]), /REMOVE_ALL) + '_' + STRCOMPRESS(STRING(date[2]), /REMOVE_ALL) + '_' + STRCOMPRESS(STRING(date[3]), /REMOVE_ALL) + '.tif'
    temp_img = READ_TIFF(temp_img_fn)
    temp_img = ROTATE(temp_img, 2)
    temp_img = HIST_EQUAL(temp_img, MINV=230, MAXV=280)
    new_img = CONGRID(temp_img, 475, 261)
    temp_list.add, new_img
ENDFOR
RETURN, temp_list
END

S.13 LOAD_WASP_IMGS

FUNCTION LOAD_WASP_IMGS, path, season

IF season EQ 0 THEN BEGIN
    date_arr = [[2, 24, 2009, 12, 0, 0], [2, 16, 2009, 13, 0, 0], [3, 4, 2009, 13, 0, 0]]
ENDIF ELSE BEGIN
    date_arr = [[2, 11, 2010, 12, 0, 0], [2, 11, 2010, 21, 0, 0], [3, 4, 2010, 15, 0, 0], [3, 4, 2010, 21, 0, 0]]
ENDELSE

s = SIZE(date_arr, /DIMENSIONS)
num_days = s[1]
wasp_list = LIST()
FOR i=0, num_days-1 DO BEGIN
  date = date_arr[*, i]
  wasp_img_fn = path + STRCOMPRESS(STRING(date[0]), /REMOVE_ALL) + 
     '_' + STRCOMPRESS(STRING(date[1]), /REMOVE_ALL) + 
     '_' + STRCOMPRESS(STRING(date[2]), /REMOVE_ALL) + 
     '_' + STRCOMPRESS(STRING(date[3]), /REMOVE_ALL) + 
     '_lwir_small.png'
  wasp_img = READ_PNG(wasp_img_fn)
  wasp_list.add, wasp_img
ENDFOR
RETURN, wasp_list

S.14 MAKE_ALGE_IMAGE

PROCEDURE MAKE_ALGE_IMAGE, path, date, temp_img_data, ice_img_data,
   params, psfc, ICE=ice, WATER=water
   row_loc_below = WHERE(STRMATCH(params, 'NX (# OF NODES IN X-DIR)') EQ 1)
   num_rows = FLOAT(params[row_loc_below-1])
   num_rows = num_rows[0]
   col_loc_below = WHERE(STRMATCH(params, 'NY (# OF NODES IN Y-DIR)') EQ 1)
   num_cols = FLOAT(params[col_loc_below-1])
   num_cols = num_cols[0]
   hours_loc_below = WHERE(STRMATCH(params, 'TMAX (TOTAL RUN TIME, HOURS)') EQ 1)
   num_hours = FLOAT(FIX(params[hours_loc_below-1]))
   num_hours = num_hours[0]
   extracted = STRSPLIT(psfc[1], /EXTRACT)
   first_date = extracted[9]
   first_time = extracted[10]
   first_date = FLOAT(STRSPLIT(first_date, '/', /EXTRACT))
   first_time = FLOAT(STRSPLIT(first_time[0], ':', /EXTRACT))
   beg_time = JULDAY(first_date[0], first_date[1], first_date[2], first_time[0],
      first_time[1], 0)
   end_time = JULDAY(date[0], date[1], date[2], date[3], date[4], date[5])
   total_time = (end_time-beg_time)*(2.4/0.1)
   first_line = num_rows*total_time
   last_line = first_line + (num_rows-1)
   IF KEYWORD_SET(water) THEN BEGIN
      temp_img_fn = path + 'tempimg' + STRCOMPRESS(STRING(date[0]), /REMOVE_ALL)
         + '_' + STRCOMPRESS(STRING(date[1]), /REMOVE_ALL) + 
         '_' + STRCOMPRESS(STRING(date[2]), /REMOVE_ALL) + '
         _lwir_small.png'
      temp_img_arr = FLTARR(1)
      temp_img_arr[0] = 999.99
      FOR i=first_line, last_line DO BEGIN
temp_data = STRSPLIT(temp_img_data[i], /EXTRACT)
    temp_img_arr = [temp_img_arr, temp_data]
ENOIF
    temp_img = temp_img_arr[1:*]
    temp_img = REFORM(temp_img, num_cols, num_rows)
WRITE_TIFF, temp_img_fn, temp_img, /FLOAT
ENDIF

IF KEYWORD_SET(ice) THEN BEGIN
    ice_img_fn = path + 'iceimg' + STRCOMPRESS(STRING(date[0]), /REMOVE_ALL) + ' ' + STRCOMPRESS(STRING(date[1]), /REMOVE_ALL) + ' ' + STRCOMPRESS(STRING(date[2]), /REMOVE_ALL) + ' ' + STRCOMPRESS(STRING(date[3]), /REMOVE_ALL) + '.tif'
    ice_img_arr = FLTARR(1)
    ice_img_arr[0] = 999.99
FOR i=first_line, last_line DO BEGIN
    ice_data = STRSPLIT(ice_img_data[i], /EXTRACT)
    ice_img_arr = [ice_img_arr, ice_data]
ENDFOR
    ice_img = ice_img_arr[1:*]
    ice_img = REFORM(ice_img, num_cols, num_rows)
WRITE_TIFF, ice_img_fn, ice_img, /FLOAT
ENDIF
END

S.15 MAKE_AVG_FLOWS

PRO MAKE_AVG_FLOWS; flow_fn
flow_fn = DIALOG_PICKFILE()
nlines_flow = FILE_LINES(flow_fn)
flow = STRARR(nlines_flow)
OPENR, unit, flow_fn, /GET_LUN
READF, unit, flow
FREE_LUN, unit
flow = FLOAT(flow)
dir = FILE_DIRNAME(flow_fn)
base = FILE_BASENAME(flow_fn, '.dat')
num_params = 20.0
window_size = nlines_flow/num_params
flow_avg = FLTARR(nlines_flow)
avg_params = FLTARR(num_params)
avg_params_fn = dir + '/' + base + '_avgparams.dat'
OPENW, unit, avg_params_fn, /GET_LUN
FOR i=0, num_params-1 DO BEGIN
    data_block = MEAN(flow[i*window_size:(i+1)*window_size-1])
    PRINTF, unit, STRING(STRCOMPRESS(data_block, /REMOVE_ALL))
    flow_avg[i*window_size:(i+1)*window_size-1] = data_block
ENDFOR
FREE_LUN, unit
avg_fn = dir + '/' + base + '_avg.dat'
OPENW, unit, avg_fn, /GET_LUN
FOR i=0, nlines_flow-1 DO BEGIN
PRINTF, unit, STRING(STRCOMPRESS(flow_avg[i],/REMOVE_ALL))
ENDFOR

p = PLOT(flow_avg, 'r', YRANGE = [-5, 8])
FREE_LUN, unit
END

S.16 METRIC_ENGINE

FUNCTION METRIC_ENGINE, ALGE_cube, PTS_arr, num_days, alge_path, new_data_path, metric_flag
    ratio_ice = METRIC_ICE(alge_path, new_data_path, metric_flag)
    observed = FLTARR(1)
    simulated = FLTARR(1)
    all_xpts = FLTARR(1)
    all_ypts = FLTARR(1)
    FOR i=0, num_days-1 DO BEGIN
        ; PTS TO WARP is the coordinates in ALGE space that need to be warped
        ; using an affine matrix
        ; and then compared to their associated WASP points to calculate
        ; difference between the ALGE and WASP imagery
        pts_fn = PTS_arr[i]
        nlines_pts = FILE_LINES(pts_fn)
        pts_strarr = STRARR(nlines_pts)
        OPENR, unit, pts_fn, /GET_LUN
        READF, unit, pts_strarr
        FREE_LUN, unit
        LUT = FLTARR(3,nlines_pts)
        FOR cur_line=0L, nlines_pts-1 DO BEGIN
            extracted = STRSPLIT(pts_strarr[cur_line], /EXTRACT)
            LUT[*, cur_line] = FLOAT(extracted)
        ENDFOR
        xpts = LUT[0, *]
        ypts = LUT[1, *]
        all_xpts = [all_xpts, REFORM(xpts)]
        all_ypts = [all_ypts, REFORM(ypts)]
        ALGE_slice = REFORM(ALGE_cube[i, *, *])
        ALGE_slice_big = CONGRID(ALGE_slice, 4748, 2606)
        observed_day = REFORM(LUT[2, *]) + 273.15
        simulated_day= ALGE_slice_big[REFORM(LUT[0, *]), REFORM(LUT[1, *])]
        observed = [observed, observed_day]
        simulated = [simulated, simulated_day]
    ENDFOR
observed = observed[1:*
simulated = simulated[1:*
all_xpts = all_xpts[1:*
all_ypts = all_ypts[1:*
good_inds = WHERE(simulated NE 0.0)
observed = observed[good_inds]
simulated = simulated[good_inds]
all_xpts = all_xpts[good_inds]
all_ypts = all_ypts[good_inds]
num_pts = N_ELEMENTS(observed)
ratio_water = ALGE_METRIC(simulated, observed, metric_flag)
observed_str = STRING(observed)
simulated_str = STRING(simulated)
title_arr = ['Simulated', 'Observed']
water_ratios_fn = new_data_path + 'water_ratios.dat'
OPENW, unit, water_ratios_fn, /GET_LUN
PRINTF, unit, new_data_path
PRINTF, unit, ' XPT YPT ' + title_arr[0] + ' ' + title_arr[1]
FOR i=0, num_pts-1 DO BEGIN
  PRINTF, unit, STRING(all_xpts[i]) + ' ' + STRING(all_ypts[i]) + ' ' + STRING(simulated_str[i]) + ' ' + STRING(observed_str[i])
ENDFOR
PRINTF, unit, 'Water ratio: ' + STRING(ratio_water)
FREE_LUN, unit
RETURN, [ratio_ice, ratio_water]
END

FUNCTION METRIC_ICE, alge_path, new_data_path, metric_flag
fraction_data_fn = FILE_WHICH(alge_path, 'ice_fraction_data.txt')
param_fn = FILE_WHICH(alge_path, 'param.dat')
nlines_iceFrac = FILE_LINES(fraction_data_fn)
iceFrac_strarr = STRARR(nlines_iceFrac)
OPENR, unit, fraction_data_fn, /GET_LUN
READF, unit, iceFrac_strarr
FREE_LUN, unit
nlines_param = FILE_LINES(param_fn)
params = STRARR(nlines_param)
OPENR, unit, param_fn, /GET_LUN
READF, unit, params
FREE_LUN, unit
row_loc_below = WHERE(STRMATCH(params, 'NX (# OF NODES IN X-DIR)') EQ 1)
num_rows = FLOAT(params[row_loc_below-1])
num_rows = num_rows[0]
col_loc_below = WHERE(STRMATCH(params, 'NY (# OF NODES IN Y-DIR)') EQ 1)
num_cols = FLOAT(params[col_loc_below-1])
num_cols = num_cols[0]

iceFrac_arr = STRARR(5)
FOR cur_line=0, nlines_iceFrac-1 DO BEGIN
extracted = STRSPLIT(iceFrac_strarr[cur_line], /EXTRACT)
    iceFrac_arr = [[iceFrac_arr], [extracted]]
ENDFOR

iceFrac_arr = iceFrac_arr[*, 1:*]
observed_ice_fracs = FLTARR(nlines_iceFrac)
simulated_ice_fracs = FLTARR(nlines_iceFrac)

FOR i=0, nlines_iceFrac-1 DO BEGIN
    fn = 'iceimg' + STRCOMPRESS(STRING(iceFrac_arr[0, i]), /REMOVE_ALL) + '_' +
    STRCOMPRESS(STRING(iceFrac_arr[1, i]), /REMOVE_ALL) + '_' +
    STRCOMPRESS(STRING(iceFrac_arr[2, i]), /REMOVE_ALL) + '_' +
    STRCOMPRESS(STRING(iceFrac_arr[3, i]), /REMOVE_ALL) + '.tif'
    ice_img_fn = new_data_path + fn
    ice_img = READ_TIFF(ice_img_fn)
    simulated_ice_fracs[i] = CALC_ICE_COVERAGE(ice_img, new_data_path)
    observed_ice_fracs[i] = iceFrac_arr[4, i]
ENDFOR

; Calculate ratio metric for ice coverage
ratio_ice = ALGE_METRIC(simulated_ice_fracs, observed_ice_fracs, metric_flag)
str_simulated = STRING(simulated_ice_fracs)
str_observed = STRING(observed_ice_fracs)
title_arr = ['Month', 'Day', 'Year', 'Hour', 'Simulated', 'Observed']

ice_ratios_fn = new_data_path + 'ice_ratios.dat'
OPENW, unit, ice_ratios_fn, /GET_LUN
FOR i=0, nlines_iceFrac-1 DO BEGIN
PRINTF, unit, STRING(iceFrac_arr[0, i]) + ' ',
    STRING(iceFrac_arr[1, i]) + ' ',
    STRING(iceFrac_arr[2, i]) + ' ',
    STRING(iceFrac_arr[3, i]) + ' ',
    STRING(str_simulated[i]) + ' ',
    STRING(str_observed[i])
ENDFOR

PRINTF, unit, 'Ice ratio: ' + STRING(ratio_ice)
FREE_LUN, unit
RETURN, ratio_ice
END

FUNCTION RUN_MAKE_ALGE_IMAGE, season, new_data_path, alge_path, ICE=ice, WATER=water
temp_img_fn = FILE_WHICH(new_data_path, 'fort.32')
ice_img_fn = FILE_WHICH(new_data_path, 'fort.53')
sfc_file_fn = FILE_WHICH(new_data_path, 'sfc.dat')
param_fn = FILE_WHICH(new_data_path, 'param.dat')
fraction_data_fn = FILE_WHICH(alge_path, 'ice_fraction_data.txt')

ice_imgs = 0
temp_imgs = 0
pice_img_data = 0

IF KEYWORD_SET(water) THEN BEGIN
    nlines_temp = FILE_LINES(temp_img_fn)
temp_img_data = STRARR(nlines_temp)
OPENR, unit, temp_img_fn, /GET_LUN
READF, unit, temp_img_data
FREE_LUN, unit
ptemp_img_data = PTR_NEW(temp_img_data)
temp_imgs = 1
ENDIF

IF KEYWORD_SET(ice) THEN BEGIN
    nlines_ice = FILE_LINES(ice_img_fn)
tice_img_data = STRARR(nlines_ice)
OPENR, unit, ice_img_fn, /GET_LUN
READF, unit, ice_img_data
FREE_LUN, unit
pice_img_data = PTR_NEW(ice_img_data)
ice_imgs = 1
ENDIF

nlines_param = FILE_LINES(param_fn)
params = STRARR(nlines_param)
OPENR, unit, param_fn, /GET_LUN
READF, unit, params
FREE_LUN, unit

nlines_iceFrac = FILE_LINES(fraction_data_fn)
ticeFrac_strarr = STRARR(nlines_iceFrac)
OPENR, unit, fraction_data_fn, /GET_LUN
READF, unit, ticeFrac_strarr
FREE_LUN, unit

; Open meteorology file and read in weather data to string array
nlines_sfc = FILE_LINES(sfc_file_fn)
sfc = STRARR(nlines_sfc)
OPENR, unit, sfc_file_fn, /GET_LUN
READF, unit, sfc
FREE_LUN, unit

date_arr = INTARR(6, nlines_iceFrac)
FOR i=0, nlines_iceFrac-1 DO BEGIN
    extracted = STRSPLIT(ticeFrac_strarr[i], /EXTRACT)
date_arr[*], i = [FIX(extracted[0]), $ month
    FIX(extracted[1]), $ day
    FIX(extracted[2]), $ year
    FIX(extracted[3]), $ hour
    0, $ minutes
    0] $ seconds
ENDFOR
s = SIZE(date_arr, /DIMENSIONS)
num_days = s[1]

FOR i=0, num_days-1 DO BEGIN
  PRINT, date_arr[*, i]
  MAKE_ALGE_IMAGE, new_data_path, $
  date_arr[* , i], $
  temp_img_data, $
  ice_img_data, $
  params, $
  sfc, $
  ICE=ice_imgs, $
  WATER=temp_imgs
ENDFOR

IF KEYWORD_SET(water) THEN PTR_FREE, ptemp_img_data
IF KEYWORD_SET(ice) THEN PTR_FREE, pice_img_data
RETURN, 1
END

FUNCTION RUN_METRIC_ENGINE, season, alge_path, new_data_path, metric_flag

; Season=0 means 08/09 winter
; Season=1 means 09/10 winter

IF season EQ 0 THEN BEGIN
  num_days = 3
  img_2_16_ALGE_FN = new_data_path + 'tempimg2_16_2009_13.tif'
  img_2_24_ALGE_FN = new_data_path + 'tempimg2_24_2009_12.tif'
  img_3_4_ALGE_FN = new_data_path + 'tempimg3_4_2009_13.tif'
  img_2_16_ALGE = READ_TIFF(img_2_16_ALGE_FN)
  img_2_24_ALGE = READ_TIFF(img_2_24_ALGE_FN)
  img_3_4_ALGE = READ_TIFF(img_3_4_ALGE_FN)
  s_alge = SIZE(img_2_16_ALGE, /DIMENSIONS)
  num_col_alge = s_alge[0]
  num_row_alge = s_alge[1]
  ALGE_cube = FLTARR(num_days, num_col_alge, num_row_alge)
  ALGE_cube[0, *, *,] = img_2_16_ALGE
  ALGE_cube[1, *, *,] = img_2_24_ALGE
  ALGE_cube[2, *, *,] = img_3_4_ALGE
  PTS_arr = [alge_path + '02162009LUT.dat', $
              alge_path + '02242009LUT.dat', $
              alge_path + '02302009LUT.dat']
266
CHAPTER S. PSO-ALGE Code

... ENDIF ELSE BEGIN...

num_days = 4

day_2_11_ALGE_FN = new_data_path + 'tempimg2_11_2010_12.tif'
night_2_11_ALGE_FN = new_data_path + 'tempimg2_11_2010_21.tif'
day_3_4_ALGE_FN = new_data_path + 'tempimg3_4_2010_16.tif'
night_3_4_ALGE_FN = new_data_path + 'tempimg3_4_2010_21.tif'

... day_2_11_ALGE = READ_TIFF(day_2_11_ALGE_FN) ...
night_2_11_ALGE = READ_TIFF(night_2_11_ALGE_FN)
day_3_4_ALGE = READ_TIFF(day_3_4_ALGE_FN)
night_3_4_ALGE = READ_TIFF(night_3_4_ALGE_FN)

s_alge = SIZE(day_2_11_ALGE, /DIMENSIONS)
num_col_alge = s_alge[0]
num_row_alge = s_alge[1]

ALGE_cube = FLTARR(num_days, num_col_alge, num_row_alge)
ALGE_cube[0, *, *] = day_2_11_ALGE
ALGE_cube[1, *, *] = night_2_11_ALGE
ALGE_cube[2, *, *] = day_3_4_ALGE
ALGE_cube[3, *, *] = night_3_4_ALGE

PTS_arr = [alge_path + '02112010_1_LUT.dat',
           alge_path + '02112010_2_LUT.dat',
           alge_path + '03042010_1_LUT.dat',
           alge_path + '03042010_1_LUT.dat']

... ENDELSE ...

ratio_array = METRIC_ENGINE(ALGE_cube, PTS_arr, num_days, alge_path,
                             new_data_path, metric_flag)

RETURN, ratio_array

END

S.20 READ_ALL_ALGE_DATA

FUNCTION RUN_METRIC_ENGINE, season, alge_path, new_data_path, metric_flag

; Season=0 means 08/09 winter
; Season=1 means 09/10 winter

IF season EQ 0 THEN BEGIN

num_days = 3

img_2_16_ALGE_FN = new_data_path + 'tempimg2_16_2009_13.tif'
img_2_24_ALGE_FN = new_data_path + 'tempimg2_24_2009_12.tif'
img_3_4_ALGE_FN = new_data_path + 'tempimg3_4_2009_13.tif'

img_2_16_ALGE = READ_TIFF(img_2_16_ALGE_FN)
img_2_24_ALGE = READ_TIFF(img_2_24_ALGE_FN)
img_3_4_ALGE = READ_TIFF(img_3_4_ALGE_FN)

s_alge = SIZE(img_2_16_ALGE, /DIMENSIONS)
num_col_alge = s_alge[0]
num_row_alge = s_alge[1]

ALGE_cube = FLTARR(num_days, num_col_alge, num_row_alge)
ALGE_cube[0, *, *] = img_2_16_ALGE
ALGE_cube[1, *, *] = img_2_24_ALGE
ALGE_cube[2, *, *] = img_3_4_ALGE

PTS_arr = [alge_path + '02162009LUT.dat', $
  alge_path + '02242009LUT.dat', $
  alge_path + '03042009LUT.dat']

ENDIF ELSE BEGIN

num_days = 4

day_2_11_ALGE_FN = new_data_path + 'tempimg2_11_2010_12.tif'
night_2_11_ALGE_FN = new_data_path + 'tempimg2_11_2010_21.tif'

day_3_4_ALGE_FN = new_data_path + 'tempimg3_4_2010_16.tif'
night_3_4_ALGE_FN = new_data_path + 'tempimg3_4_2010_21.tif'

s_alge = SIZE(day_2_11_ALGE, /DIMENSIONS)
num_col_alge = s_alge[0]
num_row_alge = s_alge[1]

ALGE_cube = FLTARR(num_days, num_col_alge, num_row_alge)
ALGE_cube[0, *, *] = day_2_11_ALGE
ALGE_cube[1, *, *] = night_2_11_ALGE
ALGE_cube[2, *, *] = day_3_4_ALGE
ALGE_cube[3, *, *] = night_3_4_ALGE

PTS_arr = [alge_path + '02112010_1_LUT.dat', $
  alge_path + '02112010_2_LUT.dat', $
  alge_path + '03042010_1_LUT.dat', $
  alge_path + '03042010_1_LUT.dat']

ENDELSE

ratio_array = METRIC_ENGINE(ALGE_cube, PTS_arr, num_days, alge_path, new_data_path, metric_flag)

RETURN, ratio_array

END
Appendix T

PSO-ALGE Analysis Tools

T.1 SWARM_RESULTS

```idl
PRO SWARM_RESULTS, DATA_PATH=data_path, 
   ALGE_PATH=alge_path, 
   RES_DIR=res_dir, 
   SEASON=season, 
   EXTRACT=extract, 
   RESTORE=restore, 
   SAVE_FILE=save_file, 
   ORIG_FLOW=orig_flow

COMPILE_OPT idl2

IF N_ELEMENTS(season) EQ 0 THEN season=0
IF N_ELEMENTS(res_dir) EQ 0 THEN PRINT, 'Results directory required' && RETURN

IF KEYWORD_SET(extract) THEN BEGIN
   IF N_ELEMENTS(data_path) EQ 0 THEN PRINT, 'Extraction requires DATA_PATH'
      && RETURN
   IF N_ELEMENTS(alge_path) EQ 0 THEN PRINT, 'Extraction requires ALGE_PATH'
      && RETURN
   IF N_ELEMENTS(save_file) EQ 0 THEN save_file = res_dir + 'big_data.sav'
   RAW_DATA_EXTRACTION, ALGE_PATH=alge_path, 
      DATA_PATH=data_path, 
      RESULTS_DIR=res_dir, 
      SAVE_FILE=save_file
ENDIF

IF KEYWORD_SET(restore) THEN RESTORE, save_file

IF KEYWORD_SET(orig_flow) THEN BEGIN
   CD, res_dir
   truth_fn = res_dir + 'flow.dat'
   nlines_truth = FILE_LINES(truth_fn)
   truth = STRARR(nlines_truth)
   OPENC, unit, truth_fn, /GET_LUN
   READF, unit, truth
```

269
FREE_LUN, unit
truth_avg = FLOAT(truth)
window_size = nlines_truth/num_parameters

ENDIF

CD, res_dir
best_particle = WHERE(all_scores[num_actual_gens-1, *] EQ_MIN(all_scores[
    num_actual_gens-1, *]))
best_particle_flow = all_pbest[num_actual_gens-1, *, best_particle]
first_best_particle_flow = all_pbest[0, *, best_particle]
best_plotting = FLTARR(nlines_truth)
first_plotting = FLTARR(nlines_truth)
FOR j=0, num_parameters-1 DO BEGIN
    block1 = REPLICATE(best_particle_flow[j], window_size)
    block2 = REPLICATE(first_best_particle_flow[j], window_size)
    best_plotting[j*window_size:(j+1)*window_size-1] = MEAN(block1)
    first_plotting[j*window_size:(j+1)*window_size-1] = MEAN(block2)
ENDFOR

pos = [0.66,1.08]
PRINT, 'Making pbest plots'
pbest_plot = PLOT(truth_avg, '2', YRANGE=[0, ub+1], XTITLE='Time [hours]',
    YTITLE='Flow rate [$m^3/s$]', TITLE=plot_title, DIMENSIONS=[500, 500], BUFFER=1, NAME='Actual
Flow Avg.')
pbest_plot.YRANGE = [0, ub+1]
pbest_plot.XRANGE = [0, N_ELEMENTS(truth_avg)-1]
pbest_plot.FONT_SIZE = 14
pbest_plot.FONT_STYLE = 0
pbest_plot.COLOR = 'steel blue'

p = PLOT(best_plotting, '2', COLOR='firebrick', OVERPLOT=pbest_plot, NAME='Particle Flow')
l = LEGEND(TARGET=[pbest_plot, p], POSITION=pos, /NORMAL, ORIENTATION=1,
    LINESTYLE=0, FONT_SIZE=8, SHADOW=0, /RELATIVE)
fn = 'pbest_final.pdf'
pbest_plot.SAVE, fn, BITMAP=0, PAGE_SIZE=[5,5]
pbest_plot.CLOSE

pbest_1_plot = PLOT(truth_avg, '2', YRANGE=[0, ub+1], XTITLE='Time [hours]',
    YTITLE='Flow rate [$m^3/s$]', TITLE=plot_title, DIMENSIONS=[500, 500], BUFFER=1, NAME='Actual
Flow Avg.')
pbest_1_plot.YRANGE = [0, ub+1]
pbest_1_plot.XRANGE = [0, N_ELEMENTS(truth_avg)-1]
pbest_1_plot.FONT_SIZE = 14
pbest_1_plot.FONT_STYLE = 0
pbest_1_plot.COLOR = 'steel blue'
p1 = PLOT(first_plotting, '2', COLOR='firebrick', YRANGE=[lb, ub], OVERPLOT=
pbest_1_plot, NAME='Particle Flow')
l = LEGEND(TARGET=[pbest_1_plot, p1], POSITION=pos, /NORMAL, ORIENTATION=1,
T.1. SWARM_RESULTS

LINESTYLE=0, FONT_SIZE=8, SHADOW=0, /RELATIVE)

fn = 'pbest_first.pdf'
pbest_1_plot.SAVE, fn, BITMAP=0, PAGE_SIZE=[5,5]
pbest_1_plot.CLOSE

ice_sim_first = FLTARR(nlines_ice_first-3)
ice_obs_first = FLTARR(nlines_ice_first-3)

ice_sim_final = FLTARR(nlines_ice_final-3)
ice_obs_final = FLTARR(nlines_ice_final-3)

FOR i=2, (nlines_ice_first-2) DO BEGIN
vals_ice_1 = STRSPLIT(ice_first[i], /EXTRACT)
  ice_sim_first[i-2] = vals_ice_1[4]
  ice_obs_first[i-2] = vals_ice_1[5]

vals_ice_2 = STRSPLIT(ice_final[i], /EXTRACT)
  ice_sim_final[i-2] = vals_ice_2[4]
  ice_obs_final[i-2] = vals_ice_2[5]
ENDFOR

iceP1_fn = 'ice_first_corr.pdf'
iceP2_fn = 'ice_final_corr.pdf'

PLOT_CORRELATIONS, ice_obs_first, ice_sim_first, iceP1_fn, /ICE
PLOT_CORRELATIONS, ice_obs_final, ice_sim_final, iceP2_fn, /ICE

PRINT, 'Work with swarm scores'
first_score_fn = 'first_scores.dat'
OPENW, unit, first_score_fn, /GET_LUN
PRINTF, unit, all_scores[0, *]
FREE_LUN, unit

final_score_fn = 'final_scores.dat'
OPENW, unit, final_score_fn, /GET_LUN
PRINTF, unit, all_scores[num_actual_gens-1, *]
FREE_LUN, unit

PRINT, 'Make first and final flow plots'
first_flows = REFORM(all_pbest[0, *, *])
final_flows = REFORM(all_pbest[num_actual_gens-1, *, *])

IF KEYWORD_SET(orig_flow) THEN BEGIN
PRINT, 'Make actual flow plots'
first_avg_fn = 'first_actual_flows.pdf'
final_avg_fn = 'final_actual_flows.pdf'
PLOT_FLOWS, first_flows, truth_avg, nlines_truth, window_size, first_avg_fn
  , lb[0], ub[0], /VARIABLE
PLOT_FLOWS, final_flows, truth_avg, nlines_truth, window_size, final_avg_fn
  , lb[0], ub[0], /VARIABLE

first_fn = 'first_flows.pdf'
final_fn = 'final_flows.pdf'
PLOT_FLOWS, first_flows, truth_avg, nlines_truth, window_size, first_fn, lb
  [0], ub[0], /ALL
272  

CHAPTER T. PSO-ALGE Analysis Tools

138 PLOT_FLOWS, final_flows, truth_avg, nlines_truth, window_size, final_fn, lb [0], ub[0], /ALL
139 
140 ENDIF ELSE BEGIN
141 PRINT, 'Make average flow plots'
142 first_avg_fn = 'first_avg_flows.pdf'
143 final_avg_fn = 'final_avg_flows.pdf'
144 PLOT_FLOWS, first_flows, truth_avg, nlines_truth, window_size, first_avg_fn , lb[0], ub[0], /AVERAGE
145 PLOT_FLOWS, final_flows, truth_avg, nlines_truth, window_size, final_avg_fn , lb[0], ub[0], /AVERAGE
146 
147 first_fn = 'first_flows.pdf'
148 final_fn = 'final_flows.pdf'
149 PLOT_FLOWS, first_flows, truth_avg, nlines_truth, window_size, first_fn, lb [0], ub[0], /ALL
150 PLOT_FLOWS, final_flows, truth_avg, nlines_truth, window_size, final_fn, lb [0], ub[0], /ALL
151 ENDELSE
152
153
154 ; PRINT, 'Making flow frames'
155 ; FILE_MKDIR, res_dir + 'flow_frames/'
156 ; CD, res_dir + 'flow_frames'
157 ; FOR i=0, num_actual_gens-1 DO BEGIN
158 ; flow = REFORM(all_pbest[i,*,*])
159 ; pbest_flow = REFORM(all_pbest[i,*, best_particle])
160 ; print_gen = STRCOMPRESS(STRING(i), /REMOVE_ALL)
161 ; flow_fn = 'gen' + print_gen + '.png'
162 ; pbest_fn = 'pbest' + print_gen + '.png'
163 ; PLOT_FLOWS, flow, truth_avg, nlines_truth, window_size, flow_fn, lb[0], ub[0]
164 ; PLOT_FLOWS, pbest_flow, truth_avg, nlines_truth, window_size, pbest_fn, lb [0], ub[0], /SINGLE
165 ; ENDFOR
166
167 END

T.2 PLOT_CORRELATIONS

1 PRO PLOT_CORRELATIONS, x, y, fn, ICE=ice, WATER=water, FLOW=flow
2 
3 IF KEYWORD_SET(ice) THEN BEGIN
4 x_range = [0.0, 1.0]
5 y_range = [0.0, 1.0]
6 x_title = 'Observed Ice Fractions'
7 y_title = 'Simulated Ice Fractions'
8 ENDIF
9 IF KEYWORD_SET(water) THEN BEGIN
10 x_range = [MIN(x), MAX(x)]
11 y_range = [MIN(y), MAX(y)]
12 x_title = 'Observed Water Temperatures [K]'
13 y_title = 'Simulated Water Temperatures [K]'
14 ENDIF
15 IF KEYWORD_SET(flow) THEN BEGIN
x_range = [MIN(x), MAX(x)]
y_range = [MIN(y), MAX(y)]
x_title = 'Observed Flow Rates [m$^3$/s]'
y_title = 'Simulated Flow Rates [m$^3$/s]'
ENDIF

pos = [0.67, 0.95]
plot_pos = [0.52, 0.45]

p1 = PLOT(x, y, '1', COLOR='steel blue', SYMBOL='o', LINESTYLE=6, XRANGE=x_range,
          YRANGE=y_range, XTITLE=x_title, YTITLE=y_title, $
          TITLE=plot_title, DIMENSIONS=[500, 500], BUFFER=1, FONT_SIZE=14,
          FONT_STYLE=0, NAME='Ice fractions')

p1.SYM_FILLED=1
p1.POSITION=plot_pos

; Calculate the Pearson correlation coefficient.
coefficient = CORRELATE(x, y, /DOUBLE)
params = LINFIT(x, y, /Double, YFIT=yfit)
numPerfectPts = 100.
x_perf = DINDGEN(numPerfectPts+1) / numPerfectPts * (1.00 - 0.0) + 0.0
p2 = PLOT(x, yfit, '-2', COLOR='firebrick', OVERPLOT=p1, NAME='Data correlation')

p3 = PLOT(x_perf, (1*x_perf+0), '--2', COLOR='sea green', OVERPLOT=p1, NAME='One
to-one line')
t2 = TEXT(0.6, 0.17, /DATA, '$R^2$ = ' + STRING(coefficient, Format='(F0.3)'),
          FONT_STYLE=1, FONT_SIZE=10)
t2 = TEXT(0.6, 0.1, /DATA, 'y = ' + STRING(params[1], Format='(F0.2)'),
          FONT_STYLE=1, FONT_SIZE=10)

l = LEGEND(TARGET=[p2, p3], POSITION=pos, /NORMAL)
l.POSITION = pos
l.LINESTYLE = 0
l.FONT_SIZE = 14
l.SHADOW = 0
l.ORIENTATION=0

p1.SAVE, fn, BITMAP=0, PAGE_SIZE=[5,5]
END

T.3 PLOT_FLOWS
pos = [0.67,1.0]
plot_pos = [0.52,0.52]

p1 = PLOT(truth_avg, '1', YRANGE=[0, ub+1], XTITLE='Time [hours]', YTITLE='Flow rate [m\textsuperscript{3}/s]', TITLE=plot_title, DIMENSIONS=[500,500], BUFFER=1, NAME='True Flow')
ax = p1.AXES
ax[2].MAJOR = 0
ax[3].MAJOR = 0
ax[2].MINOR = 0
ax[3].MINOR = 0
p1.XRANGE = [0, N_ELEMENTS(truth_avg)-1]
p1.FONT_SIZE = 14
p1.FONT_STYLE = 0
p1.POSITION = plot_pos
p1.COLOR = 'dark turquoise'
sim_avg = MAKE_ARRAY(nlines_truth, /FLOAT, VALUE= MEAN(flows))
IF KEYWORD_SET(all) THEN BEGIN
FOR i=0, num_parts-1 DO BEGIN
cur_particle = FLTARR(nlines_truth)
FOR j=0, num_params-1 DO BEGIN
cur_particle[j*window_size:(j+1)*window_size-1] = flows[j,i]
ENDFOR
p2 = PLOT(cur_particle, '1', OVERPLOT=p1, NAME='Swarm Flows')
p2.COLOR = 'indian red'
ENDFOR
p3 = PLOT(sim_avg, '--2', OVERPLOT=p1, NAME='Swarm Average')

truth_plot = MAKE_ARRAY(nlines_truth, /FLOAT, VALUE= MEAN(truth_avg))
p4 = PLOT(truth_plot, '--2', OVERPLOT=p1, NAME='True Average')

l = LEGEND(TARGET=[p1, p4, p2, p3], /NORMAL)
l.POSITION = pos
l.LINESTYLE = 0
l.FONT_SIZE = 14
l.SHADOW = 0
p2.COLOR = 'indian red'
p3.COLOR = 'firebrick'
p4.COLOR = 'steel blue'
p1.SAVE, fn, BITMAP=0, PAGE_SIZE=[5,5]
ENDIF

IF KEYWORD_SET(variable) THEN BEGIN
sd_arr = FLTARR(num_params)
avg_arr = FLTARR(num_params)
x_arr = FLTARR(num_params)
cur_particle = FLTARR(nlines_truth)
FOR i=0, num_parts-1 DO BEGIN
param_stats = MOMENT(flows[i,*], SDEV=sd, MEAN=avg)
sd_arr[i] = sd
avg_arr[i] = avg
cur_particle[1+window_size:(i+1)+window_size-1] = avg
x = [i+window_size,(i+1)+window_size-1]
x_arr[i] = MEAN(x)
ENDFOR

p2 = PLOT(cur_particle, '1', OVERPLOT=p1, NAME='Swarm Flows')
p3 = PLOT(sim_avg, '--2', OVERPLOT=p1, NAME='Swarm Average')
p4 = ERRORPLOT(x_arr, avg_arr, sd_arr, '3', OVERPLOT=p1, LINESTYLE=6)
p4.COLOR = 'grey'

truth_plot = MAKE_ARRAY(nlines_truth, /FLOAT, VALUE= MEAN(truth_avg))
p5 = PLOT(truth_plot, '--2', OVERPLOT=p1, NAME='True Average')

l = LEGEND(TARGET=[p1, p5, p2, p3], /NORMAL)
l.POSITION = pos
l.LINESTYLE = 0
l.FONT_SIZE = 14
l.SHADOW = 0
p2.COLOR = 'indian red'
p3.COLOR = 'firebrick'
p5.COLOR = 'steel blue'
p1.SAVE, fn, BITMAP=0, PAGE_SIZE=[5,5]
ENDIF

IF KEYWORD_SET(average) THEN BEGIN
sd_arr = FLTARR(num_params)
avg_arr = FLTARR(num_params)
x_arr = FLTARR(num_params)
cur_particle = FLTARR(nlines_truth)

FOR i=0, num_params-1 DO BEGIN
param_stats = MOMENT(flows[i, *], SDEV=sd, MEAN=avg)
sd_arr[i] = sd
avg_arr[i] = avg
cur_particle[1+window_size:(i+1)+window_size-1] = avg
x = [i+window_size,(i+1)+window_size-1]
x_arr[i] = MEAN(x)
ENDFOR

p2 = PLOT(cur_particle, '1', OVERPLOT=p1, NAME='Swarm Flows')
p3 = PLOT(sim_avg, '--2', OVERPLOT=p1, NAME='Swarm Average')
p4 = ERRORPLOT(x_arr, avg_arr, sd_arr, 'b3', OVERPLOT=p1, LINESTYLE=6)

l = LEGEND(TARGET=[p1, p2, p3], /NORMAL)
l.POSITION = pos
l.LINESTYLE = 0
l.FONT_SIZE = 14
l.SHADOW = 0
p2.COLOR = 'indian red'
p3.COLOR = 'firebrick'
p4.COLOR = 'steel blue'
p1.SAVE, fn, BITMAP=0, PAGE_SIZE=[5,5]

ENDIF

IF KEYWORD_SET(single) THEN BEGIN
cur_particle = FLTARR(nlines_truth)
FOR j=0, num_params-1 DO BEGIN
  cur_particle[j*window_size:(j+1)*window_size-1] = flows[j]
ENDFOR
p2 = PLOT(cur_particle, '1', OVERPLOT=p1, NAME='Particle Flow')
p2.COLOR = 'indian red'
l = LEGEND(TARGET=[p1, p2], /NORMAL)
l.POSITION = pos
l.LINESTYLE = 0
l.FONT_SIZE = 14
l.SHADOW = 0
p1.SAVE, fn, BITMAP=0, PAGE_SIZE=[5,5]
ENDIF
END