



**US Army Corps
of Engineers®**
Engineer Research and
Development Center



Coastal Field Data Collection Program

Continuously Operating Dune-Mounted Lidar System at the Field Research Facility

A Report Detailing Lidar Collection, Processing, Evaluation, and Product Development

Katherine L. Brodie, A. Tristan Dyer, Nicholas J. Spore,
Richard K. Slocum, Annika M. O'Dea, E. Tucker Whitesides,
and Alexander D. Renaud

November 2018



The U.S. Army Engineer Research and Development Center (ERDC) solves the nation's toughest engineering and environmental challenges. ERDC develops innovative solutions in civil and military engineering, geospatial sciences, water resources, and environmental sciences for the Army, the Department of Defense, civilian agencies, and our nation's public good. Find out more at www.erdclibrary.usace.army.mil.

To search for other technical reports published by ERDC, visit the ERDC online library at <http://acwc.sdp.sirsi.net/client/default>.

Continuously Operating Dune-Mounted Lidar System at the Field Research Facility

A Report Detailing Lidar Collection, Processing, Evaluation, and Product Development

Katherine L. Brodie, Nicholas J. Spore, and Alexander D. Renaud

Coastal and Hydraulics Laboratory

Field Research Facility

U.S. Army Engineer Research and Development Center

1261 Duck Road

Duck, NC 27949

A. Tristan Dyer and E. Tucker Whitesides

Department of Civil, Construction, and Environmental Engineering

North Carolina State University

2501 Stinson Drive

Raleigh, NC 27695

Richard K. Slocum and Annika M. O'Dea

School of Civil and Construction Engineering

101 Kearney Hall

Oregon State University

Corvallis, OR 97331

Final report

Approved for public release; distribution is unlimited.

Prepared for Coastal Field Data Collection (CFDC) Program
U.S. Army Engineer Research and Development Center
Coastal and Hydraulics Laboratory
Vicksburg, MS 39180-6199

Under Project 468430, "Remote Sensing"

Abstract

The Coastal and Hydraulics Laboratory, Field Research Facility (FRF), stationary lidar tower in Duck, NC, provides hourly, high-density point cloud data of the ocean, foreshore, and dune. This report documents the system and approaches used to collect, process, and analyze the lidar data. Data collection consists of a single 30-minute linescan and three framescans collected every hour. The linescans generate a two-dimensional cross-shore timeseries of the dry beach, swash, inner-surf zone waves, and water levels. The three framescans are used to generate a three-dimensional point-cloud and digital elevation model of the dry beach, quantifying changes along 500 meters of beach. Data are processed in near-real-time using automated algorithms that translate raw data into accurate geo-referenced coordinates by co-registering all data to fixed objects. Linescan data provide quantitative measurements of beach morphology change on very short timescales (seconds to minutes), as well as time-series statistics that include wave run-up elevations, inner-surf zone waves, and setup. This system improves upon the FRF capacity to observe the beach and surfzone onshore of the sandbar in near-real-time. These measurements will therefore serve as a crucial component of the FRF Coastal Model Testbed, providing detailed morphological and hydrodynamic observations at the shoreline.

DISCLAIMER: The contents of this report are not to be used for advertising, publication, or promotional purposes. Citation of trade names does not constitute an official endorsement or approval of the use of such commercial products. All product names and trademarks cited are the property of their respective owners. The findings of this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

DESTROY THIS REPORT WHEN NO LONGER NEEDED. DO NOT RETURN IT TO THE ORIGINATOR.

Contents

Abstract	ii
Figures and Tables.....	vi
Preface.....	viii
Unit Conversion Factors	ix
Abbreviations.....	x
1 Introduction.....	1
1.1 Background.....	1
1.2 Approach	2
1.3 Objective	3
2 System Specifications.....	5
2.1 Local environment.....	5
2.2 Tower construction	7
2.3 Lidar setup.....	9
2.4 Power and data connectivity.....	10
3 Data Collection and Initial Setup Procedures.....	12
3.1 System specifications and overview.....	12
3.2 Collection methods.....	13
3.2.1 <i>LidarCollect</i>	14
3.2.2 <i>StationaryTowerCollect.sh</i>	17
3.2.3 <i>Task scheduling</i>	18
3.3 Baseline scan	18
3.4 Initial orientation scans.....	19
3.5 Coordinate system.....	22
4 Real-Time Data Processing.....	23
4.1 Data folder structure	23
4.2 Generating the parameters file	24
4.3 Initiating automated data processing	25
4.4 Converting file formats.....	27
4.5 Rectification and coregistration.....	28
4.5.1 <i>Initial data transformation and coregistration</i>	30
4.5.2 <i>Coregistration accuracy assessment</i>	34
4.5.3 <i>Generation of final transformation matrix</i>	37
4.5.4 <i>QA/QC figure generation</i>	38
4.6 Framescan processing.....	40
4.6.1 <i>Reading and merging data</i>	40
4.6.2 <i>Filtering and saving LAS files</i>	41
4.6.3 <i>Generating bare earth DEMs</i>	43

4.6.4	Generating a QA/QC figure.....	43
4.7	Linescan processing.....	44
4.7.1	Pre-processing linescan data	45
4.7.2	Generating raw data images	50
4.7.3	Runup line extraction.....	52
4.7.4	Filtering.....	55
4.7.5	Finalizing the pre-processing.....	55
4.7.6	Scientific analysis and data product generation.....	56
4.7.7	Saving images and data	57
4.8	Reprocessing data.....	58
5	Quality Assurance and Quality Control (QA/QC).....	61
5.1	Lidar Monitor	61
5.2	RunupTool	62
5.2.1	Setup.....	63
5.2.2	Program layout.....	64
5.2.3	Users.....	65
5.2.4	Discovery	65
5.2.5	Visualizing data	67
5.2.6	Specifying existing runup line.....	69
5.2.7	Simple editing tools	70
5.2.8	Algorithmic editing tools	70
5.2.9	Saving and committing edits	72
6	Data Products	73
6.1	Linescan data products.....	73
6.1.1	Morphology.....	73
6.1.2	Wave runup	75
6.1.3	Hydrodynamics.....	77
6.1.4	Wave gauge.....	79
6.2	Framescan data products	80
6.3	Beach digital elevation model (DEM)	81
6.4	NetCDF Generation for linescan and framescan data products.....	82
7	Conclusion.....	86
	References	88
	Appendix A: Lidar Specifications.....	91
	Appendix B: Parameters File Format.....	94
	Appendix C: RXPconvert.....	100
	Appendix D: 3ddconvert	103
	Appendix E: readLidar	106
	Appendix F: Cloud-Filters	110

Appendix G: points2grid	122
Appendix H: Range Filter.....	123
Appendix I: Variable List	127
Report Documentation Page	

Figures and Tables

Figures

Figure 2-1. ERDC-CHL FRF site.....	5
Figure 2-2. Ocean-side environment viewed by the lidar tower.	6
Figure 2-3. Dune crest and landward side of the lidar tower.	6
Figure 2-4. Reflector used to rectify lidar scan data.	7
Figure 2-5. Lidar tower assembly at north end of FRF property.	7
Figure 2-6. Lidar mast and scanner supported by cradle over dune crest.	8
Figure 2-7. Lidar scanner, housing, and mounting plate connected to mast.	9
Figure 2-8. Image of mounting plate to which the lidar is attached.	9
Figure 2-9. Lidar tower control cabinet.	10
Figure 2-10. Diagram of lidar tower control cabinet.	11
Figure 3-1 Lidar tower data network configuration.	12
Figure 3-2. Plan-view of distribution of reflectors used to rectify the scanner data.	21
Figure 4-1. Flowchart of coregistration process and associated products generated.	30
Figure 4-2. Top-down view of a lidar scan (white and gray points) with distance alongshore on the horizontal axis and distance cross-shore along the vertical axis. Red boxes highlight the locations of the 11 planes used in the plane-based coregistration, labeled by number.....	32
Figure 4-3. Top-down view of a lidar scan (white and gray points) with distance alongshore on the horizontal axis and distance cross-shore along the vertical axis. Red boxes indicate locations of the three planes used in the error analysis.....	34
Figure 4-4. The coregistered lidar scan (shown in white) is analyzed for error against Planes 1, 2, and 3, (red). Plane 1 (y-z plane in FRF coordinates) is a support beam on the pier, Plane 2 (x-z plane in FRF coordinates) is the front face of the building to the immediate south of the main FRF building, and Plane 3 (an x-y plane in FRF coordinates) is part of the cul-de-sac in the neighborhood to the north of the lidar.....	35
Figure 4-5. The combined error from the plane error assessment from the rectified scans (in red) and the coregistered scans (blue) (top); the estimated translational error from the least squares adjustment (middle); and the estimated rotational errors from the least squares adjustment (bottom) from 5 September 2015 to 30 November 2015.....	37
Figure 4-6. zChange figure example from 20151008-1200 showing (1) a plot of the difference between the raw rectified scan and the baseline scan (top left); (2) a plot of the difference between the coregistered scan and the baseline scan (top right); (3) a plot of the difference between the raw rectified scan and the raw rectified scan from the previous hour (bottom left); and (4) a plot of the difference between the coregistered scan and the coregistered scan from the previous hour (bottom right).	39
Figure 4-7. DEM figure example from 20151001-0200 showing (1) a plot of the unfiltered gridded reflectance, (2) a plot of the unfiltered gridded elevation, and (3) a plot of the filtered gridded elevation with water points removed.	44
Figure 4-8 (a) The angle of the lidar laser footprint is not necessarily aligned with the FRF coordinate system. (b) Any tilt in the lidar scanner away from vertical will introduce curvature to the lidar laser footprint. (c) The cross-shore and offline distance values of a single return.	47

Figure 4-9. A sample 1D cross-shore grid that is aligned with the vector created by points P1 and P2.....	48
Figure 4-10. A cross-shore profile of all points in a raw dataset.....	51
Figure 4-11. A cross-shore profile of 10 individual scan lines from a filtered dataset.	52
Figure 4-12. The cross-shore position of wave runup (magenta) along the beach at a single point in time from Argus video imagery. The blue line represents a single cross-shore profile through time.....	53
Figure 4-13. The cross-shore position of wave runup (magenta) for a single cross-shore profile through time (e.g., along the blue line in Figure 4-12). These data are from video imagery for ease of viewing but are similar to the data captured by the lidar scanner.	53
Figure 4-14. A plot showing samples of raw data as well as morphological, hydrodynamics, and runup line data products.....	58
Figure 4-15. A plot showing the <i>zDiffCumulative</i> variable values as well as the runup line in terms of its distance down-line through time.....	58
Figure 5-1. The <i>Lidar Monitor</i> web page.	62
Figure 5-2. The <i>RunupTool</i> GUI Layout, featuring 1 – Program toolbar, 2 – editing toolbar, 3 – file viewer, 4 – variable viewer, 5 – user toolbar, 6 – plot section chooser, and 7 – plot viewer.	64
Figure 5-3. The <i>RunupTool</i> Add New User dialog.....	65
Figure 5-4. The <i>RunupTool</i> settings menu.....	66
Figure 5-5. <i>RunupTool</i> plotting options.	67
Figure 5-6. A <i>RunupTool</i> plot of the <i>fGrid</i> variable, which is the product of the interpolated foreshore grid being subtracted from the elevation grid (<i>zGrid</i>). Warmer colors represent greater differences between the variables (deeper water). This product allows for clearer visualization of the swash elevations above the dry beach.	68
Figure 5-7. The runup lines (black line) displayed on <i>RunUpTool</i> agrid (lidar return amplitude data) plots (a) before and (b) after QA/QC. Warmer colors represent greater amplitude values.	70
Figure H-1. Lidar scanner rotation against cross-section of the beach environment.	123
Figure H-2. A bird introduces bad returns into the point cloud.....	124
Figure H-3. Example moving stencil for filter. The first 10 returns stored in an array.....	125
Figure H-4. Five point stencil with points P3 through P7 are considered when testing point P8.	125

Tables

Table 6-1. Linescan NetCDF generation files – morphology.....	83
Table 6-2. Wave runup NetCDF generation files.....	83
Table 6-3. Hydrodynamics NetCDF generation files.....	84
Table 6-4. Wave gauge linescan NetCDF generation files.....	84
Table 6-5. Beach DEM framescan NetCDF generation files.	85
Table I-1. List of variables involved in lidar tower data collection and processing.	127

Preface

This report was funded by the U.S. Army Corps of Engineers (USACE) Coastal Field Data Collection (CFDC) Program, Project No. 468430. The CFDC is administered at the U.S. Army Engineer Research and Development Center (ERDC), Coastal and Hydraulics Laboratory (CHL), under the Flood Risk Management (FRM) Business Line. Mr. Mark Roupas Headquarters, USACE, was FRM Business Line Manager overseeing CFDC. Dr. Julie D. Rosati, Coastal Processes Branch (CEERD-HFC) was the Technical Director of the FRM R&D Programs. Dr. Jeffrey P. Waters, Coastal Observations and Analysis Branch (CEERD-HFA), was Program Manager of CFDC.

The work was performed by the CEERD-HFA of the Flood and Storm Protection Division (CEERD-HF), ERDC-CHL. Technical reviews and discussions of this report were provided by Mr. Patrick J. Dickhudt of ERDC-CHL and Mr. Peter J. Gadowski of ERDC-CRREL. At the time of publication of this report, Dr. Jeffrey P. Waters was Chief, CEERD-HFA, and Dr. Cary A. Talbot was Chief, CEERD-HF. The Acting Director of ERDC-CHL was Mr. Jeffrey R. Eckstein.

COL Ivan P. Beckman was the Commander of ERDC, and the Director was Dr. David W. Pittman.

Unit Conversion Factors

Multiply	By	To Obtain
degrees (angle)	0.01745329	radians
inches	0.0254	meters

Abbreviations

1D	one-dimensional
2D	two-dimensional
3D	three-dimensional
CFDC	Coastal Field Data Collection program
CHL	Coastal and Hydraulics Laboratory
cm	centimeters
dB	decibel
DEM	digital elevation model
ERDC	U.S. Army Engineer Research and Development Center
FRF	Field Research Facility
GHz	gigahertz
GLCS	Global Coordinate System
GPS	Global Positioning System
GUI	graphical user interface
Hz	hertz
kHz	kilohertz
m	meters

NaN	not a number
NAVD88	North American Vertical Datum of 1988
netCDF	Network Common Data Form
QA/QC	Quality Assurance and Quality Control
RMS	root mean square
RMSE	root mean square error
RTK	real-time-kinematic
S	seconds
SOCS	scanner's own coordinate system
SOP	scanner orientation and position
THREDDS	Thematic Real-time Environmental Distributed Data Services
YAML	human-readable data serialization language

1 Introduction

1.1 Background

The Coastal and Hydraulics Laboratory (CHL), Field Research Facility (FRF), has been collecting near-real-time oceanographic wave and current data, as well as monthly morphology surveys, in Duck, NC, for over 30 years, as funded through the Coastal Field Data Collection Program (see <https://frfdataportal.erdcdren.mil/> for access to the data). Wave observations are collected hourly and extend from the shallow surf-zone (~1 meter [m] nominal depth) out to 26 m water depth in a cross-shore array (Hanson et al. 2009) that is aligned with National Oceanic and Atmospheric Administration buoy 44014 in 47.6 m water depth, just prior to the continental shelf break. Wave data in 17 m water depth extend back to 1980, with high-resolution directional wave data beginning in 1990 in 8 m water depth from an array of pressure sensors (Long and Oltman-Shay 1991). Beach and surf-zone topographic and bathymetric surveys have been collected approximately monthly, as well as before and after many storms, since 1980 along cross-shore transects that span the 1 kilometer (km) property (Forte et al. 2017).

Beginning in 1986, daily (now half-hourly) video images of the beach and nearshore (Lippman and Holman 1989) have also been collected from the continuously operating Argus tower, which is run jointly with Oregon State University. The addition of the video imaging capability provided higher temporal frequency observations of sandbar movement (Lippmann and Holman 1990), shoreline position (Pianca et al. 2015), wave runup (Holman 1986; Stockdon et al. 2006), as well as bathymetry (Holman et al. 2013). While video imagery approaches provide quantitative morphodynamic data of the beach and surf-zone, stationary, single cameras cannot provide measurements of beach elevation (volume) changes or detailed hydrodynamic observations (sea-surface elevations) in the inner-surf and swash zones. These observations are critical for improving the U.S. Army Corps of Engineers (USACE) numerical models to accurately predict and plan for coastal change, inundation, and infrastructure damage during storms.

Accurate predictions of (1) morphological change at the shoreline and (2) inundation elevations from large waves and surge are critical to the

USACE ability to properly manage flood risks during coastal storms. Unfortunately, much of this change and the highest water levels occur during the peak of the storm, when it can be both difficult and dangerous to collect high-quality data to ground truth the numerical models. For example, rain drops can obscure views from camera lenses, surf-zone conditions are unsafe for vessels, and rapid bed elevation changes, actively breaking non-linear waves, and shoreline erosion can make maintaining in situ sensors and interpreting their data challenging. Assessing and improving models that predict the erosion and recovery of coasts during and following storms demand high-quality morphodynamic data in the inner-surf and swash zone *during*, and not just before and after, storms (Roelvink et al. 2009; van Rooijen et al. 2012; Kobayashi and Jung 2012).

Coastal storm impacts often scale with maximum storm run-up elevations, defined as the highest location that ocean water reaches on the beach (Sallenger 2000; Stockdon et al. 2007; Masselink and van Heteran 2014). The elevation of maximum wave runup can be defined statistically by an exceedance elevation over a certain period of time (Stockdon et al. 2006) and is the sum of a number of complex processes, including tides, storm surge, the increase in mean sea level due to breaking waves (wave setup), and the oscillating excursion of the wave-driven swash. The transformation of infragravity and swell-sea waves across surf-zone and shoreline bathymetry affects the magnitude of wave setup and swash, and therefore the resulting maximum wave runup. Shoreline erosion and inner surf-zone bathymetry evolution during storms can be rapid and often spatially variable. This morphologic change is a result of interactions between cross-shore undertow, alongshore currents, and oscillatory incident- and infragravity-wave-driven flows that interact to drive sediment transport (Aagaard and Greenwood 1994; Houser et al. 2006; Houser and Greenwood 2007). Accurately resolving sea-surface elevations and morphologic response at high spatial and temporal resolution in this region provides an initial step towards improving numerical models that can simulate these complex processes.

1.2 Approach

Staff at the FRF have focused on developing the application of lidar technology for observing the dynamic coastal zone (Brodie et al. 2012; Brodie et al. 2015). Lidar technology utilizes pulses of light emitted at known times and directions to make range measurements that, when combined with the known orientation and position of the lidar scanner, can

generate high-resolution three-dimensional (3D) point clouds of the environment. Terrestrial lidar has traditionally been used to measure terrain elevation data, providing hundreds of points per square meter (Spore and Brodie 2017) and allowing detailed volume change calculations (Eamer and Walker 2013). These data are often collected from mobile platforms (Spore and Brodie 2017) or from stationary tripods on specific survey intervals (days to months) (Pietro et al. 2008; Theuerkauf and Rodriguez 2012). However, recent field experiments with terrestrial lidar scanners (Blenkinsopp et al. 2010; Brodie et al. 2012; Almeida et al. 2013; Vousdoukas et al. 2014; Brodie et al. 2015) in the littoral zone have identified the ability to simultaneously measure the elevation of the beach topography and the elevation of the free surface of seawater, particularly aerated breaking waves.

Terrestrial lidar scanners can be mounted on towers above the shoreline or beach and dune and used to measure the elevation of the water surface and uncovered beach topography between wave swash events along narrow cross-shore transects. Using this approach, lidar has been shown to accurately measure the sub-aerial beach topography and swash and inner-surf zone waves and water levels in both lab and field studies (Blenkinsopp et al. 2010; Blenkinsopp et al. 2012; Almeida et al. 2013; Puleo et al. 2014; Vousdoukas et al. 2014; Brodie et al. 2015). The spatially dense observations of the sea-surface collected with the lidar allow analysis of inner-surf zone hydrodynamics, including estimates of shoreline reflection coefficients and detailed measurements of cross-shore variations in wave setup, wave asymmetry, wave skewness, and wave spectra (Brodie et al. 2015). Drawbacks to utilizing terrestrial lidar scanners for oceanographic measurements include the limits caused by short sampling ranges and discontinuous time series due to the reflectivity of water at infrared wavelengths and mounting geometries that may result in shadowing behind tall waves or small fields of view. Most prior efforts utilizing lidar to measure hydrodynamic processes have focused on deploying lidar systems for short-term field experiments (weeks to months) whereas this application is focused on developing the workflow to continuously operate a terrestrial lidar scanner in the coastal environment.

1.3 Objective

This special report (SR) documents the FRF's continuously operating dune-mounted terrestrial lidar scanner, which is designed to make hourly measurements of 3D beach topography, wave runup elevations, and inner-

surf and swash zone waves and water levels. The tower is mounted above the dune, in line with the FRF cross-shore wave and current array at the north end of the FRF property. These data extend the FRF near-real-time, hourly, observations of waves from shallow water to the shoreline while simultaneously providing observations of 3D beach topography. Similar to the rest of the FRF observations, the data are freely available in near-real-time on the Data Portal at <https://frfdataportal.erdcdren.mil/> and Thematic Real-time Environmental Distributed Data (THREDDS) Server <https://chlthredds.erdcdren.mil>.

The following chapters provide detailed information on the lidar system physical infrastructure (Chapter 2), data collection procedures (Chapter 3), real-time data processing (Chapter 4), Quality Assurance/Quality Control (QA/QC) procedures (Chapter 5), and available data products (Chapter 6). Nine appendices summarize the lidar scanner setup specifications, code file format parameters, several C++, MATLAB, and command line programs and significant scripts, a range filter, and a list of variables involved in processing.

This SR serves as both internal documentation of procedures and code for collecting and processing data as well as a guide for outside users who wish to fully understand the collection methodology and processing of the data. The hardware and methods described here reflect the state of the system and the code as of April 2017 and may be updated or amended in subsequent years to reflect advancements in the code or hardware. All of the code and wiki information referenced in this repository can be found in the internal FRF-Remote-Sensing git¹ repository. Any file paths reported in this document are thus relative to this repository. (For more information about access to this repository, please contact Dr. Katherine Brodie at Katherine.L.Brodie@usace.army.mil.)

¹ “git” is a free and open-source distributed version control system. It is widely used for tracking software changes and sharing code. <https://git-scm.com/>

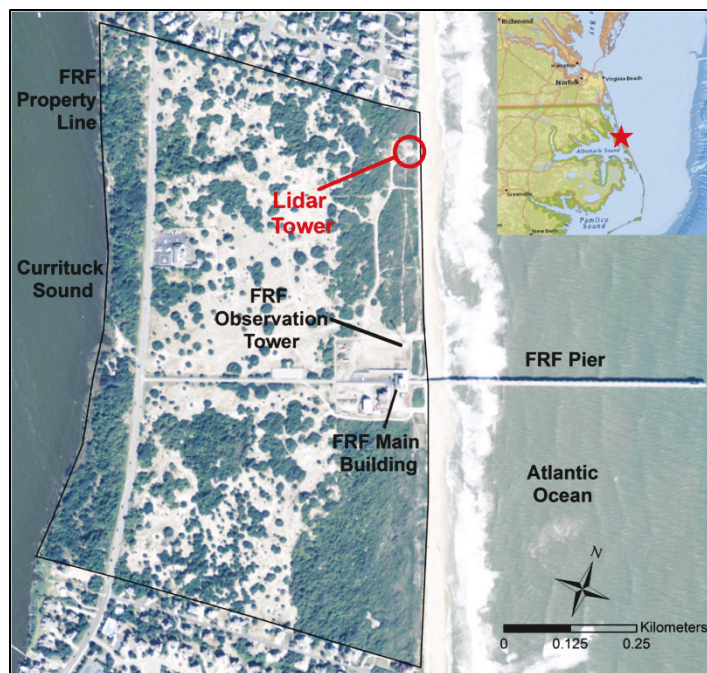
2 System Specifications

The stationary lidar tower at the FRF has been designed to provide high-density point cloud data of the ocean, foreshore, dune, and terrestrial environment for an undeveloped parcel of land on the northern FRF property. The following sections describe the environment that is visible to the lidar scanner, the tower that houses the lidar scanner, and the system configuration delivering the lidar scanner data. Specific measurements and specifications of both the tower and the lidar scanner can be found in Appendix A.

2.1 Local environment

The stationary lidar tower is located approximately 0.4 km north of the main building at the FRF (Figure 2-1). The FRF property line is located less than 0.1 km north of the lidar tower and developments including ocean-front houses extend northward, out of the range of the lidar scanner.

Figure 2-1. ERDC-CHL FRF site.



The beach at this location is typically 20–40 m wide, depending on seasonal and tidal conditions, and the average beach slope is 0.1. Tidal range is from 0.00 to 0.62 m (North American Vertical Datum of 1988

[NAVD88]) at mean low water to 0.36 m (NAVD88) at mean high water. The dune crest directly underneath the lidar scanner is approximately 6.7 m (NAVD88) high. The landward side of the dune is covered in dense vegetation, typically 0.5–1.5 m high depending on the season. Figure 2-2 and Figure 2-3 provide a view of the local environment.

Figure 2-2. Ocean-side environment viewed by the lidar tower.



Figure 2-3. Dune crest and landward side of the lidar tower.



Five permanent reflectors have been installed to geo-reference the lidar scan data each time the lidar is mounted on the tower as well as for each hourly scan, as there is temporal variability in the position and orientation of the lidar scanner due to atmospheric effects on the tower. Further details on their use are included in Section 3.4. These reflectors are distributed across the FRF property and consist of 10.5 centimeter (cm) high, 10.5 cm diameter cylinders of highly reflective material posted on steel poles of varying heights (Figure 2-4).

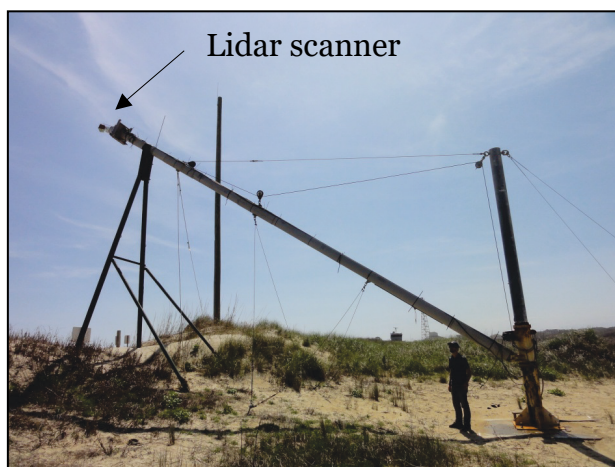
Figure 2-4. Reflector used to rectify lidar scan data.



2.2 Tower construction

The tower that supports the lidar scanner consists of an angled mast mounted to a vertical support pedestal. The lidar scanner is attached to the floating end of the angled mast that is secured by a cradle, positioning the sensor above the dune, as shown in Figure 2-5.

Figure 2-5. Lidar tower assembly at north end of FRF property.



The vertical support pedestal is anchored in a concrete pad with dimensions $2.4 \text{ m} \times 2.4 \text{ m} \times 1.5 \text{ m}$. The pedestal itself is approximately 6.5 m tall and is made of steel pipe with an outside diameter of 30 cm. The angled mast is mounted to a sleeve that contains low-friction nylon bushings, allowing the mast to rotate around the support pedestal when raised out of the cradle. The mounting point is hinged, allowing the angle of the mast to be changed using a built-in winch/pulley system. This mounting system provides easy access to the lidar scanner for

serviceability. For example, when the scanner case occasionally becomes dirty due to salty ocean spray and airborne dust and dirt, this tower design makes accessing the scanner fairly straightforward and reduces downtime when the scanner is not acquiring data. The mast can rotate up to 120 degrees (deg) counterclockwise from its operation position, and the floating end of the mast, where the lidar scanner is located, can be lowered to the ground.

The mast itself is approximately 13 m long, and when in the operating position, the floating end of the mast rests in a cradle support meant to reduce vibrations caused by high winds. The top of the cradle, shown in Figure 2-6, is located 10 m from the support pedestal and is welded to pipes that are driven 6 m into the ground. These pipes are meant to continue providing support in the event that the entire dune is washed out by a storm event. The top of the cradle support is located approximately 6 m off the ground. The mast rests in the cradle at approximately 11.7 m along the length of the mast, and it rests at an angle of approximately 30 deg above horizontal.

Figure 2-6. Lidar mast and scanner supported by cradle over dune crest.



When in operation, the mast is guyed down using four galvanized steel wires to prevent harmonic oscillations. Three 12 m telephone poles with lightning rods surrounding the system provide lightning protection for the lidar scanner (visible in Figure 2-2 and Figure 2-3).

2.3 Lidar setup

The lidar scanner itself is a Riegl VZ-1000 and is housed in a protective case, the Riegl VZ-1000-PH, seen in Figure 2-7. This protective case is mounted to the plate pictured in Figure 2-8. The protective case provides shielding from rain and wind-blown debris and particles while maintaining an optimal operating temperature with a built-in air conditioning system.

Figure 2-7. Lidar scanner, housing, and mounting plate connected to mast.



Figure 2-8. Image of mounting plate to which the lidar is attached.



The hinged mounting plate, shown in Figure 2-8, is welded to the floating end of the mast. The hinge mechanism, along with the adjustable screw spacers, allows for fine-tuning of the mounting angle, and thus the cross-shore field-of-view of the lidar scanner. This field-of-view can be adjusted

to include the entire area underneath the lidar scanner, ranging from the vertical support pedestal to the horizon above the ocean.

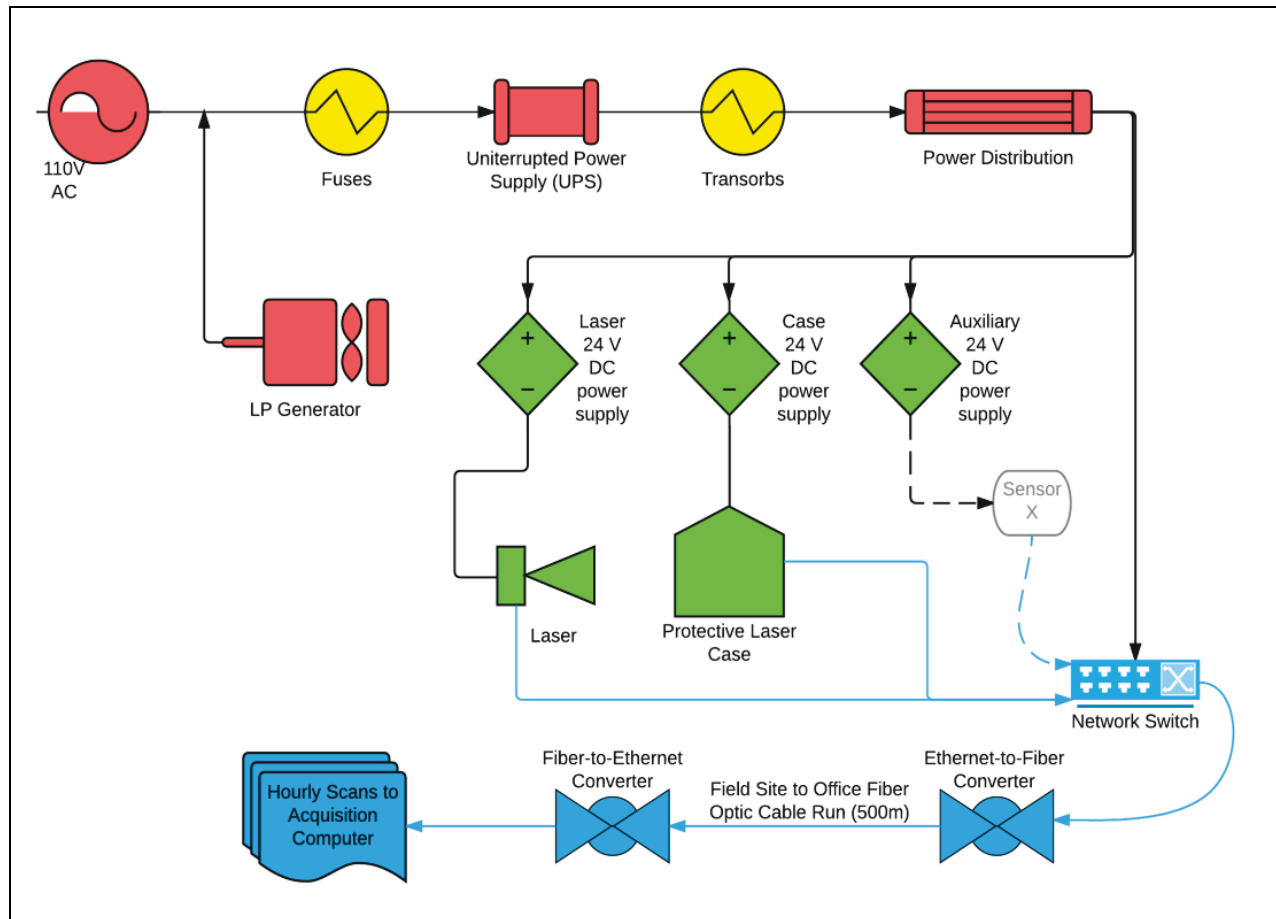
2.4 Power and data connectivity

Alternating current (AC) power is routed directly to a control cabinet (Figure 2-9) located at the base of the vertical support pedestal. An uninterruptible power supply provides power to all of the system components including two 24-volt (V) direct current (DC) power supplies (one for the scanner, one for the protective housing) and the Ethernet switch. Surge suppression and transorbs mitigate any potential power spikes that could cause damage to the equipment. A diagram depicting the power distribution and Ethernet data is shown in Figure 2-10. Note that extra power is allocated for any auxiliary sensors that may be added to the system for future work or individual short-term experiments.

Figure 2-9. Lidar tower control cabinet.



Figure 2-10. Diagram of lidar tower control cabinet.

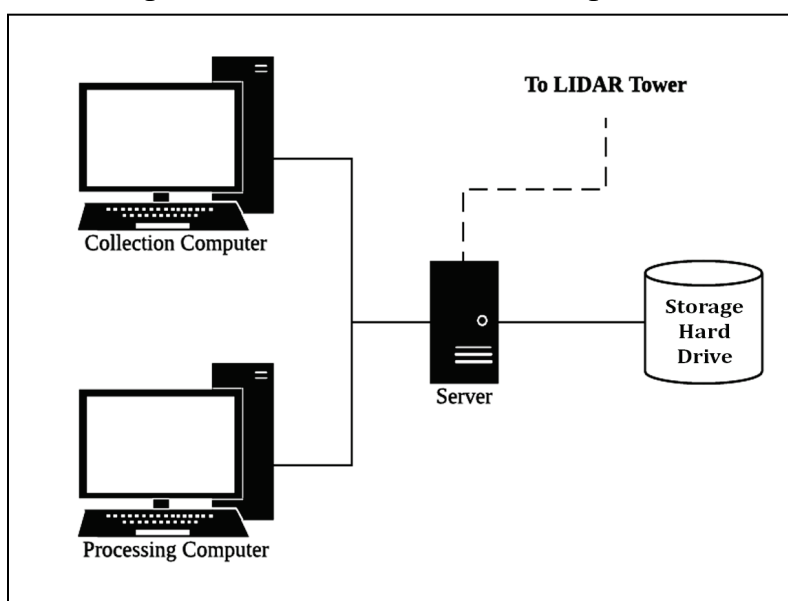


3 Data Collection and Initial Setup Procedures

3.1 System specifications and overview

The collection and processing of data from the stationary lidar system is controlled using a system of two desktop computers and a storage hard drive located on a closed network in the configuration shown in Figure 3-1.

Figure 3-1 Lidar tower data network configuration.



In this configuration, the collection computer is responsible for communicating directly with the lidar scanner and saving data locally as data are received from the scanner. Once a scan has completed, the data are transferred over the network to the storage hard drive enabled network attached storage (NAS). The processing computer monitors the NAS and processes the new data as data come in, storing the results of the data processing alongside the raw datasets on the storage hard drive. The folder storage and filing naming protocols are discussed further in Section 4.1.

While not a requirement, the separation of the data collection from the data processing onto two computers has been shown to allow for more reliable data collection, as the computer is less vulnerable to the heavy processing loads associated with the data processing.

3.2 Collection methods

The hourly lidar data collection consists of a single 30-minute linescan and three framescans that take 19 minutes to complete. The 30-minute linescan generates a time-series along a 2D cross-shore profile. The 30-minute duration was chosen to adequately sample the majority of wave-driven hydrodynamics (note that 30 minutes may undersample infragravity energy during storms) while still allowing adequate time for the completion of the subsequent framescans and the processing of the complete dataset without accumulating a backlog of data (processing of 30-minute linescan data take approximately 45 minutes to complete, including the coregistration process). Each framescan observes a different field of view, including (1) the area north of the scanner, (2) the beach and the area east of the scanner, and (3) the area south of the scanner. The fields of view include enough overlap to allow for the individual scans to be combined into a single dataset with a seamless field of view. Three separate scans, as opposed to one large framescan, were chosen to ensure a more consistent horizontal data density. For a constant scanner angular increment, the horizontal data density of rectified lidar data decreases as incident angle decreases. The time required to acquire a scan also increases as the angular stepwidth decreases. Therefore, to reduce acquisition time while increasing horizontal data density, the northernmost and southernmost portions of the beach are acquired with slower, higher-resolution frame scans. The data in the region closest to the lidar are acquired with a larger angle stepwidth, which scans faster and still acquires data with a similar horizontal data density. This sampling methodology also reduces the cumulative file size.

The data collection computer is an 8 Core Intel i7-4790 central processing unit at 3.60 gigahertz (GHz) with the Ubuntu 14.04 operating system. It communicates with both the lidar scanner and the storage hard drive over a wired local area network connection. The scanner itself is controlled by a custom program called *LidarCollect*, which leverages the Riegl C++ API to set lidar parameters and start/stop data acquisition. A bash script called *StationaryTowerCollect.sh* is responsible for setting the parameters for the three frame scans and the line scan, and when called sequentially, initializes collection of each scan. Datasets are collected hourly by using the task scheduler to execute the *StationaryTowerCollect.sh* on a regular schedule. Both the *LidarCollect* program and *StationaryTowerCollect.sh* script are located in the FRF Remote Sensing git repository and are described in the following sections.

3.2.1 LidarCollect

The *LidarCollect* program, written in C++, provides a command-line interface for communicating with the VZ series of scanners from Riegl. It allows users to query the scanner for information, set scanner parameters, and perform scans with various levels of logging and command-line output. System requirements, compiling instructions, and complete documentation of all command-line options are available in the wiki of the FRF-Remote-Sensing git repository, *LidarCollect*, repository [/FRF-Remote-Sensing/LidarCollect/wiki](#) .

One setting for the VZ series of lidar scanners that is essential to acquiring data of the water surface cannot be selected using Riegl's C++ Application Programming Interface and must be adjusted either through the graphical user interface (GUI) on the scanner itself or through the web-browser interface of the scanner. This setting is termed "Sensitivity Optimization," and must be turned "OFF" to allow for digitization of low-amplitude returns off of the water surface. The internal algorithms for the VZ series of lidar scanners have a more aggressive filter than the Z series of lidar scanners (Brodie et al. 2015), which causes points on the water surface with low amplitude to not be recorded by default. The "Sensitivity Optimization" setting reduces the effect of the aggressive filter and allows the lower amplitude water points to be returned. The setting also allows other low-amplitude returns off of particulates in the atmosphere to add noise to the data that must be filtered out later.

3.2.1.1 Linescans

The command used to perform the 30-minute linescan, which can be found in the *StationaryTowerCollect.sh* bash script, is similar to the following (with IP addresses and file paths changed):

```
./LidarCollect --ip 192.168.0.13 --line 0 30 130 0.025 180.2 1800  
FRFNProp --dir /path/to/output_dir --log --status  
/path/to/status.txt --warnings /path/to/warnings.txt
```

The flags and options used in the command are described as follows:

- `--ip 192.168.0.13` – sets the IP address of the lidar scanner
- `--line 0 30 130 0.025 180.2 1800` – sets the parameters for a single linescan. These parameters, in order, are the following:
 - **program** – the measurement program, as defined by the VZ-1000 documentation, one of
 - 0: 1400m (70 kHz)
 - 1: 1200m (100 kHz)
 - 2: 950m (150 kHz)
 - 3: 450m (300 kHz)
 - 4: Reflector
 - **θ_{start}** – the starting vertical angle
 - **θ_{end}** – the ending vertical angle
 - **$\theta_{\text{increment}}$** – the vertical angle stepwidth
 - **ϕ** – the horizontal angle (direction of the scan)
 - **duration** – the duration of the scan in seconds
 - **file identifier** – an optional string to be included in the scan's filename
- `--dir /path/to/output_dir` – the full path to the directory in which to save the raw data
- `--log` – enable output of log messages to the command-line
- `--status /path/to/status.txt` – save all scanner status messages to the specified text file
- `--warnings /path/to/warnings.txt` – save all scanner warning messages to the specified text file

Using this command, the *LidarCollect* program will connect to the scanner located at 192.168.0.13, orient the scanner head to a horizontal angle of 180.2 deg, and perform a 30-minute linescan using a peak laser pulse repetition rate of 70 kHz, which allows for a maximum measurement range of approximately 1,400 m. The linescan field of view ranges from 30 to 130 deg and will have an angular resolution of 0.025 degree. This setup is the default collection parameters for FRF data with the VZ1000 on the tower described in Section 2.2.

3.2.1.2 Framescans

The three individual framescan acquisitions are controlled via sequential calls to the *LidarCollect* program within the *StationaryTowerCollect.sh* bash script. The syntax for each is similar to the following (with IP addresses and file paths changed):

```
./LidarCollect --ip 192.168.0.13 --frame 0 63 130 0.01 256 302
0.05 FRFNProp --dir /path/to/output_dir --log --status
/path/to/status.txt --warnings /path/to/warnings.txt
```

```
./LidarCollect --ip 192.168.0.13 --frame 0 30 130 0.04 125 259
0.04 FRFNProp --dir /path/to/output_dir --log --status
/path/to/status.txt --warnings /path/to/warnings.txt
```

```
./LidarCollect --ip 192.168.0.13 --frame 0 37 130 0.01 65 128
0.05 FRFNProp --dir /path/to/output_dir --log --status
/path/to/status.txt --warnings /path/to/warnings.txt
```

The flags and options used in the commands are described as follows:

- `--ip 192.168.0.13` – sets the IP address of the lidar scanner
- `--frame 0 63 130 0.01 256 302 0.05 FRFNProp` – sets the parameters for a single framescan. These parameters, in order, are the following:
 - **program** – the measurement program, as defined by the VZ-1000 documentation, one of the following:
 - 0: 1400m (70 kHz)
 - 1: 1200m (100 kHz)
 - 2: 950m (150 kHz)
 - 3: 450m (300 kHz)
 - 4: Reflector
 - ϑ_{start} – the starting vertical angle
 - ϑ_{end} – the ending vertical angle
 - $\Delta\vartheta_{\text{increment}}$ – the vertical angle stepwidth
 - φ_{start} – the starting horizontal angle
 - φ_{end} – the ending horizontal angle
 - $\Delta\varphi_{\text{increment}}$ – the horizontal angle stepwidth
 - **file identifier** – an optional string to be included in the scan's filename

- `--dir /path/to/output_dir` – the full path to the directory in which to save the raw data
- `--log` – enable output of log messages to the command-line
- `--status /path/to/status.txt` – save all scanner status messages to the specified text file
- `--warnings /path/to/warnings.txt` – save all scanner warning messages to the specified text file

These commands are very similar to those used to perform the linescan, with the exception that the horizontal starting and ending angles and the horizontal angular stepwidth are defined. These values define the rotation of the scanner head and thus the horizontal field of view for the scan. The command-line flags and options used to define logging and file locations are identical. The values used for each of the three individual framescans performed at the FRF are those included in the example commands above.

All of the files are saved with the time-stamp of their collect in Coordinated Universal Time (UTC). For example, the linescans are saved as follows:

20170123-1400-01.FRFNProp.line.rxp

and the three framescans are saved as as follows:

20170123-1430-16.FRFNProp.frame.rxp

20170123-1436-03.FRFNProp.frame.rxp

20170123-1441-19.FRFNProp.frame.rxp

3.2.2 StationaryTowerCollect.sh

A bash script called *StationaryTowerCollect.sh*, available in the *LidarCollect* repository, is used to automate the collection of all three framescans and the linescan and store the raw data on the storage hard drive. Within the script file, a user can set the variables that define paths to directories and status files, the IP address of the scanner, the location of the *LidarCollect* executable, and the datestring format for the directory that will contain the raw data. The 30-minute linescan and three framescans are grouped into an hourly folder in the */Raw* directory on the storage hard drive for proper organization.

Before collecting, the script will ensure that the scanner is not currently in use and that the local directory that will temporarily store the raw data is valid. While an attempt is made to reduce any interruptions to the normal data acquisition, occasionally, the scanner will be manually controlled to acquire data over a specific site of interest or to improve the registration. If the scanner is currently in use at the preprogrammed time for automated acquisition, the collection will be skipped. When the scanner is not currently in use, the user-defined datestring and the current time are used to define a new folder that is created to store the raw data locally. The *LidarCollect* program is set to store the raw data in the newly created folder, and the data collection is initiated. Once data collection has completed, the script ensures that the storage hard drive is properly mounted and copies the full dataset to it before deleting the local copy. If the storage hard drive is not available for any reason, the temporary copy of the raw data will remain on the local machine. Enabling these checks into the data copying methodology ensures that no data are lost while archiving them to the storage hard drive.

3.2.3 Task scheduling

To automate the data collection, the Ubuntu task scheduler is set to call the *StationaryTowerCollect.sh* script every hour. When defining the schedule for any set of collections, it is important to be cognizant of how long each set of collections will take. For example, the collection at the FRF, consisting of a 30-minute linescan and three framescans takes approximately 45–50 minutes to complete. In this case, scheduling a collect more frequently than 45 minutes would result in every other collection being skipped.

3.3 Baseline scan

Before real-time data processing can begin, a baseline scan must be performed. The baseline scan is the benchmark from which all other subsequent scans are co-registered (Section 4.5) and therefore requires high accuracy. The baseline scan was conducted during favorable atmospheric conditions (no rain or fog and moderate humidity) and calm winds to minimize sensor movement within the scan. The scan occurred close in time to a high-accuracy Global Positioning System (GPS) survey of the reflectors used to geo-reference the scan and was performed using the same scanner parameters used in the standard real-time data collection routine.

The baseline scan is first geo-referenced using RiScanPro and the surveyed targets with GPS coordinates. The geo-referenced point cloud is read into *CloudCompare*, a free, GUI-based lidar point cloud viewing and analysis program (available at <http://www.danielgm.net/cc/>) where the data can be viewed in 3D and areas of interest can be easily identified. *CloudCompare* is used to extract coregistration regions, locations within the scan that have fixed objects that are non-changing such as walls and rooflines, that can be used to adjust subsequent scans to match the baseline's position and orientation (described in Section 4.5). Within *CloudCompare*, these coregistration regions can be defined as the dimensions of a rectangular prism (length, width, height) and the centroid of the prism (x, y, z). Regions should be well distributed around the scanner and throughout the scan and selected to maximize the along-shore and cross-shore baselines of the scan. Once defined, the full-size baseline scan is trimmed to only account for these regions. The remainder of the scan that includes sediment and vegetation is subject to real change (which is measured) and is not included as reference within the coregistration algorithm.

3.4 Initial orientation scans

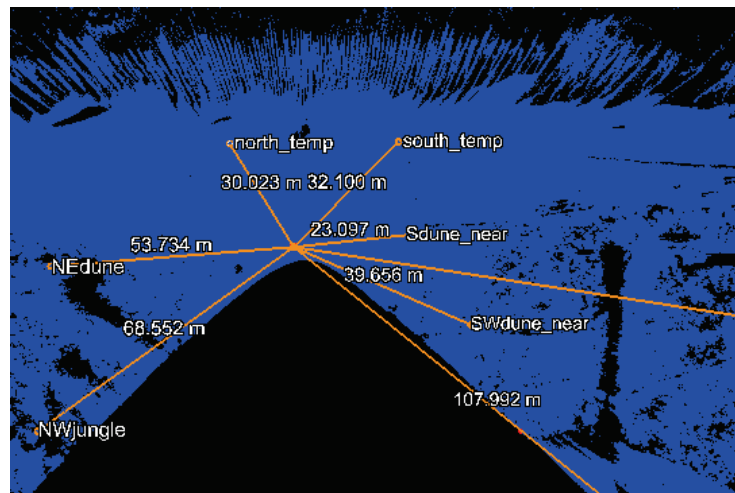
Each time the tower is raised and lowered, a new orientation and position matrix must be generated as the scanner will never return to the exact physical location as the previous deployment due to subtle imperfections in the tower's material properties and mounting tolerances. A guy wire or bolt that is tightened slightly differently between acquisitions can induce a very small angular difference in the resultant lidar position, which can propagate to significant elevation differences at distance. The raw data returned from the scanner are referenced to the scanner's own coordinate system (SOCS). To orient the data into a global coordinate system (GLCS), known tie point locations must be scanned in high detail to properly rectify the data. This process will be described in more detail in Section 4.5. A mixture of permanently installed reflectors, such as seen in Figure 2-3, and additional temporarily installed reflectors serves as the tie points.

The reflectors are surveyed with real-time-kinematic (RTK) GPS surveying equipment, each occupied for at least 3 minutes and averaged to account for and minimize horizontal and vertical error (accuracy +/- 0.025 m). The RTK-GPS surveyed reflector centers are converted into FRF coordinates (a local coordinate system, Section 3.5) and are imported into Riegl's RiScan Pro software as the GLCS points, which will be used throughout the following process. To scan these reflectors in high resolution, a 360 degree

framescan is performed, which is used to identify their respective locations within the scanner's theta (vertical) and phi (horizontal) angular coordinates. Within the 360 degree scan, the user can identify the location of highly reflective material in the framescan, and perform a subsequent *fine scan* of each reflector. The fine scan collects several thousand data points on each reflector so that the exact center of the reflector can be defined in SOCS.

Once all of the reflectors are scanned and their centroids are extracted, Riegl's "find corresponding points" tool is used to identify a rotation matrix to convert from SOCS to GLCS. This tool performs a least-squares fit to solve the most probable rotation and translation parameters to rigidly transform the scanners coordinates to global coordinates. This tool is run with a maximum error tolerance of 0.10 m and a minimum number of matched reflectors to 4. The resulting calculation reports the number of matched points, the standard deviation of the residuals, and the resulting scanner orientation and position (SOP) matrix. The goal for the user is to maximize the number of matched reflectors while minimizing the standard deviation of residuals. Once the optimal solution has been obtained, the SOP is locked from any additional modification and archived within the master RiScan Pro position and orientation project for the tower system. The tie point links can be visualized (Figure 3-2) with the respective distances from the scanner and is useful for a final quality control check on the rectification solution. The resulting SOP is exported as a *.dat* file and stored to be used as the rectification matrix outlined in Rectification and coregistration. An additional transformation matrix for the project orientation and position (POP) is also exported as a *.dat* file. This matrix defines the orientation and position of the project's coordinate system (PRCS) within the RiScan project relative to the GLCS. RiScanPro utilizes a *project coordinate system* to reduce computation time while visualizing large 3D point clouds. In the present case, the PRCS is set equal to the GLCS because the reflector centers are input in a local coordinate system (see Section 3.5), and the POP matrix is merely the identity matrix.

Figure 3-2. Plan-view of distribution of reflectors used to rectify the scanner data.



Once the transformation matrices have been exported, a MATLAB structure containing all position and orientation data, as well as the supporting metadata supplied by RiScan Pro's rectification procedure, must be created. A MATLAB script located at

DuneLidar/Setup/popsop2rot.m

and is executed using an interactive GUI. It initially prompts the user to navigate to the resulting POP.dat and SOP.dat files that were generated above. The following prompt requires the user to input metadata describing the rectification. The following information must be input (defaults for the FRF tower are listed below):

- Vertical Datum – North American Vertical Datum of 1988 (NAVD88)
- DateTime Format – MATLAB datenum of GPS time
- Time Zone – UTC
- Project Filename – path of RiScan Pro project used for generating POP and SOP .dat files
- GCP Registration Error – standard deviation of residuals reported within RiScan Pro's find corresponding points tool
- Number of GCPs – number of reflectors/corresponding points used for the SOP solution (also reported within RiScan Pro's find corresponding points tool)
- GCPs used – line number ID of the reflectors used within the RTK-surveyed reflector .xyz file (e.g., 1,2,4,6,7,8)
- Scanner ID – VZ1000

- Site ID – FRF
- Notes – any additional notes describing the rectification procedure.

After all metadata are input, an additional dialog prompts the user for the location of the .xyz file that contains the RTK-surveyed reflector centroids. The final dialog prompts the user to define the path for save the .rot.mat file. This should be the location of the *Rot* folder in the data structure outlined in Section 4.1.

3.5 Coordinate system

Due to the obliquity of the shoreline orientation relative to a north-south regular rectilinear grid, all data are rectified into a cross-shore/alongshore horizontal coordinate system known as local FRF coordinates. This system positions the data cross-shore on the x -axis and alongshore on the y -axis. The latitude and longitude (or state plane easting and northing) coordinates of reflector centers are rotated and translated into the local FRF coordinates prior to the generation of the rotation matrices (Section 3.4) using the following values:

- Origin Latitude: 36.1775975°
- Origin Longitude: -75.7496860°
 - Horizontal Datum: WGS84
 - Meters per degree of latitude: 110963.357 m
 - Meters per degree of longitude: 89953.364 m
 - Angle FRF to Lat/Lon: 71.8535°
- Origin Easting (NC state plane, meters): 901951.680 m
- Origin Northing (NC state plane, meters): 274093.1562 m
 - Horizontal Datum: NAD83 (2011)
 - Angle FRF to State Grid: 69.9747° Real-time Data Processing

4 Real-Time Data Processing

The processing of data collected from the stationary lidar tower takes place in near real-time on a dedicated processing computer. The initial setup of the processing workflow begins with a baseline scan (discussed above) and the creation of a parameters file (discussed below in Section 4.2). Once set up, a program monitors the storage hard drive for new datasets and initiates the processing workflow when a new dataset is discovered. Once processing is complete, the data products are saved to the storage hard drive and sent to the FRF Data Portal for display as hourly, non-quality-controlled data products.

An algorithm on the processing computer also monitors the storage hard drive for data that have been quality controlled by an FRF staff. These data have already been processed in near real-time and are QA/QC'd as described in Section 5. If the QA/QC process identifies any inaccuracies in the automated processing, the data are reprocessed taking these changes into account and using the same processing workflow as the raw datasets. Once reprocessing is complete, the data products are saved to the storage hard drive and sent to the FRF Data Portal for display as quality-controlled data products.

The data processing setup and workflow are written in MATLAB and make use of three utility programs, each written in C++. The entire workflow is described in the following sections.

4.1 Data folder structure

The following folder structure is used internally on the storage hard drive at the FRF to organize data as the data are collected, rectified, processed, analyzed, and QA/QC'd:

- `Baselines` – contains the baseline scan folders used in the coregistration procedures
- `FieldLog` – contains `.txt` files of any issues that arise within the physical infrastructure either hardware or software
- `GCPs` – contains `.xyz` files of the RTK-surveyed reflector centroids
- `Monthly` – contains the monthly Network Common Data Form (netCDF) files that are pushed to the FDIF server
- `Parameters` – contains all setup parameters used for processing

- `Raw` – contains the hourly folders for all the data collected with the system
- `YYYYMMDD-HHMM-SS.siteID` – each hour contains the raw `.rxp` files collected as well as all of the `.mat` and `.jpg` files generated during processing
- `RieglProjects` – contains the RiScan Pro projects used to interact with the scanner during the rectification procedure and scanning reflectors
- `Rot` – contains the POP and SOP `.dat` files from each rectification as well as the `.rot.mat` files used for processing
- `RunupTool` – contains the temporary files used within the QA/QC process to properly orient the maximum runup line over the time-series.

4.2 Generating the parameters file

The first step in setting up the automated data processing workflow, after a baseline scan has been performed, is to create a parameters file. The parameters file is a `.mat` file containing a number of required and optional variables that are used by the processing code. For example, variables defining the locations of executables, variables controlling how raw data are filtered, and variables defining the data products to be produced are all included in the parameters file. A complete listing of required and optional variables, and their effects on the processing workflow, is included in Appendix B.

The parameters file can either be generated manually or automatically by a script included in the processing code repository. The script must be edited to include file paths for the computer on which it is being run, as well as variables edited to properly suit the data that will be processed. The script is located at

DuneLidar/Setup/params/generateParamsFile.m

and can be run from within MATLAB by calling

generateParamsFile('/path/to/newParams.m');

This script generates a parameters file called *newParams.m* that can be used in the real-time data processing.

4.3 Initiating automated data processing

Once the baseline scan has been completed and the parameters file has been created, the real-time processing can begin. A MATLAB script located at

DuneLidar/MasterProcess.m

is the entry point to the entire real-time processing workflow. At the beginning of this script, the following variables are defined:

- `allDataFolder` – the path to the directory on the storage hard drive that receives the raw data from the collection computer
- `rotFolder` – the path to the directory that contains rotation matrices generated from baseline scans
- `baselineFolder` – the path to the directory that contains raw baseline scans
- `paramsFolder` – the path to the directory that contains parameters files
- `webFolder` – an optional directory that will receive copies of the images generated by the current processing dataset, useful for updating a website that provides status updates of the system
- `monthlyDir` – the path to the directory that will receive the netCDF files of quality controlled data products
- `reprocessingQueue` – the path to the text file that contains the list of datasets that have been quality controlled and are ready for reprocessing
- `numFrameScans` – the number of individual frame scans that are acquired in each complete dataset (three for the FRF)
- `numLineScans` – the number of individual linescans that are acquired in each complete dataset (one for the FRF)

The processing scripts expect the data in the `allDataFolder`, `rotFolder`, `baselineFolder`, and `paramsFolder` directories to have specific layouts and naming conventions, allowing for consistent and reliable data discovery. The `allDataFolder` expects folders of hourly scan data, as described in Section 4.1. The `rotFolder` expects a directory of the SOP and POP matrices described in Section 3.4. The `baselineFolder` should contain a folder named with a date string that contains the baseline scan data and co-registration regions described in Section 3.3. The `paramsFolder` contains a listing of params files

described in Section 4.2, named with a start date. Additionally, examples of these naming conventions and directory structures can be found in the Appendices.

Running the *MasterProcess.m* script begins the real-time processing workflow by indefinitely monitoring the raw data directory, defined by the variable `allDataFolder`, for new data, quality-controlled data, and backlogged data. Data are considered new only if created *after* the *MasterProcess.m* script has started monitoring. Data that are collected by the lidar system *before* the data processing begins are considered backlog, meaning that while they have not yet been processed, they are no longer considered *real-time* data. This categorization is an important distinction, as the real-time data take priority over backlog data due to the need to present the most up-to-date data products on the data portal. Data that have been quality controlled and are ready for reprocessing each have a line of data in the `reprocessingQueue` file. Adding a dataset to this queue is the responsibility of the tool used to perform the quality control, as explained in Section 5.

Monitoring of the raw data directory occurs in an infinite `while` loop, and at each loop the following logic is used to process the appropriate dataset:

1. A variable, `last_processed`, will contain the collection date of the last *real-time* dataset that was processed. If none have been processed yet, it will contain the date that the *MasterProcess.m* script was started.
2. Sort all datasets in the raw data directory by date.
3. Find the *newest* dataset, if it was collected *after* the date contained in the variable `last_processed`, process it as real-time data.
4. If no dataset was found for processing in step 3, check the `reprocessingQueue` file for data. Remove the first line from the reprocessing queue and reprocess that dataset as quality-controlled data.
5. If no dataset was found for processing in step 4, find the *newest* backlog dataset that was collected *before* the date contained in the variable `last_processed` and process it as real-time data.
6. If no dataset was found for processing in step 5, do nothing.
7. Loop.

The *MasterProcess.m* script will loop through this logic indefinitely until a user manually stops the processing.

4.4 Converting file formats

Once the *MasterProcess.m* script determines that a dataset must be processed, the raw data must be converted into a format that is useable by MATLAB. Because there are numerous individual files associated with a dataset, the conversion of a file into a useable format typically occurs immediately before the data in that file are needed. A script called *readLidar.m* is employed to perform these conversions. The *readLidar.m* script is located in the Utilities repository at

Utilities/Lidar/readLidar.m

and is used in the coregistration, framescan, and linescan processing steps, as described in the next three sections.

To date, two models of lidar scanner have been used at the FRF to collect data. The first is the VZ line of scanners from Riegl (most frequently, the VZ-1000), which all produce *.*rxp* files as raw data, and the second is the Riegl Z390, which produces *.*3dd* files as raw data. Both of these file formats are binary and proprietary, and to work in an automated setting, two programs were created to convert these files into a useable format. The *RXPconvert* program is used to convert *.*rxp* files, and the *3ddconvert* program is used to convert *.*3dd* files. Each program has a number of options for which data to extract from the raw data files, as well as numerous options for output formats. Both programs are included in the FRF Remote Sensing code repository and are described in depth in the Appendices C and D, respectively. Note that both require proprietary libraries from Riegl to work correctly.

The *readLidar* function, described in depth in Appendix E, takes a path to a raw data file, as well as a number of optional arguments, and reads the lidar data from the raw data file into MATLAB. The optional arguments include which returns to read from the file, which data to read from each return (e.g. position, GPS time, return intensity, etc.), the location of the *RXPconvert* or *3ddconvert* executable, and others.

4.5 Rectification and coregistration

The first step in data processing is to calculate a transformation matrix that can be used to rotate and translate framescan and linescan data into accurate geo-referenced coordinates. The rectification matrix that is determined each time the tower is raised must be adjusted each hour to reflect any small movements in the position and orientation of the scanner from hour to hour. These small movements are commonly attributed to thermal expansion in the lidar tower, which slightly shifts the position and orientation of the lidar. This section will describe the methodology used to perform the initial rectification and coregistration of the lidar data to determine an orientation matrix that can be used to translate both the framescan and linescan data into accurate geo-referenced coordinates. Here, rectification refers to the process of orienting the data in a known coordinate system with the original orientation matrix whereas coregistration refers to the process of aligning two datasets of the same region so that any georeferencing errors that have resulted from slight movements in the position and orientation of the lidar scanner are minimized.

In the context of the FRF data processing workflow, this process is accomplished using the *Coregister.m* script, located at

DuneLidar/Coregistration/Coregister.m

This script takes the following as arguments:

- `dataFolder` – path to the directory containing raw data
- `rotFolder` – path to the directory containing rotation matrices (as *.mat* files)
- `baselineFolder` – path to the directory containing baseline scans
- `paramFile` – path to the parameters file
- `savecoreg` – boolean flag indicating whether or not to save coregistration information in a **.coreg.mat* file.

In its current state, the geo-referencing process for each hour includes three steps. In the first step, the scan is rectified into FRF coordinates and then coregistered using a least squares plane-matching approach. The coregistration algorithm first trims the scan to only include points on 11 planes in the scan. Covariance matrices generated from the propagation of raw measurement error (including range, angle, and laser beam width) to

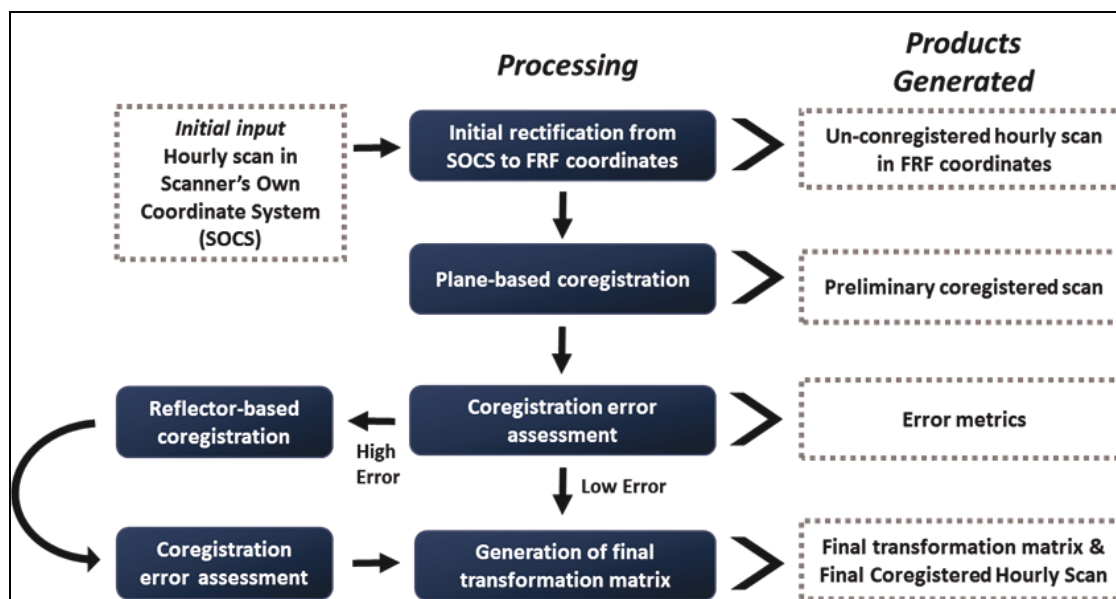
the coordinates of each point are used to weight the points in the scan being coregistered. These points are then conditioned to lie on the control plane surface (the same planes from the baseline scan, described in Section 3.3) through the adjustment of the six parameters of a rigid body transform. The computed covariance from least squares adjustment algorithm is used to assess the accuracy of the coregistration.

In some cases, the point density in the hourly scan is too low to successfully perform coregistration using the plane-matching approach. Low point densities occur during periods of rain or fog, which can attenuate the light, or when the face of the lidar case becomes dirty. If the plane-matching process fails, a least squares fit of five permanent reflectors in the scan view is used as an alternative method to coregister the scan. For this approach, the coregistration algorithm first finds the location of five permanent reflectors in the scan (described in Section 2.1) and then minimizes the distance between the reflector centroid in the current scan and the reflector centroid in the baseline scan using a least squares adjustment. The coregistration process is repeated a second time using only the four closest reflectors. The reflector-based method and plane-based method both generate a coregistration matrix.

In the second step, the accuracy of these coregistration matrices is assessed by finding the root mean square error (RMSE) in the location of three two-dimensional (2D) assessment planes in the scan relative to their locations in the baseline scan. In scans coregistered using the plane-matching approach, this error assessment is conducted and saved in addition to the estimated error generated during the least-squares adjustment process. For scans that necessitated coregistration using the reflector locations, the RMSE of the assessment plane locations is determined after the scan has been transformed using (1) the coregistration matrix found using the plane-matching technique, (2) the coregistration matrix found using all five reflectors, (3) the coregistration matrix found using only four reflectors, (4) the coregistration matrix from the previous hour, and (5) the rectification matrix. The errors are then compared between cases, and the coregistration matrix for the case that produces the smallest error is selected.

In the final step, a figure is produced to visually assess the coregistration accuracy. Each of these steps (summarized in Figure 4-1) will be described in more detail in the following sections.

Figure 4-1. Flowchart of coregistration process and associated products generated.



4.5.1 Initial data transformation and coregistration

First, the three framescans are loaded and merged into a single dataset using the *readLidar.m* script described in the previous section. The script then uses the appropriate rectification matrix, found in the `rotFolder` directory, to translate and rotate the raw data into a known local coordinate system. At the FRF, this matrix transforms the raw data from the scanner's own coordinate system to the FRF Coordinate System. The appropriate rectification matrix is dependent on the time and date of the scan being coregistered and is determined using the *getFirstFileBefore.m* script. Next, the scan is trimmed to only include points on the 11 planes in the scan using the *findAndTrimDataRegions.m* function. This function is located at

DuneLidar/Coregistration/Functions/findAndTrimDataRegions.m

The function requires the *xyz* point cloud from the scan and the dimensions (*x*-, *y*-, and *z*-distances) and center (*x*-, *y*-, and *z*-coordinates) of the control regions (in this case, the control regions are the 11 planes used in the coregistration process, located in the *parameters* file) as input arguments. The function produces an *xyz* pointcloud that includes all points in the scan that fall within the defined *x*-, *y*-, and *z*-ranges of the control region as well as a vector of numeric labels indicating in which control regions each point falls. The 11 planes used in the coregistration process are shown in Figure 4-2. After the scan has been trimmed to only

include these planes, coordinate covariance matrices are calculated using the *PropRangeEncoderBeam.m* function, located in

DuneLidar/Coregistration/Functions/ PropRangeEncoderBeam.m

The function requires the *xyz* point cloud as the input argument and produces a covariance matrix used to weight points in the least squares adjustment. A least squares fit is then completed using the *Rigid3DTransform_Planes.m* function, located in

DuneLidar/Coregistration/Functions/ Rigid3DTransform_Planes.m

This function requires the baseline planes structure (located in the *parameters* file), the planar points in the scan being coregistered, the covariance matrix generated in the previous step, the initial rotation values, and the initial translation values as input arguments. The function solves the nonlinear least squares problem such that the most probable three rotation parameters ($\theta_1, \theta_2, \theta_3$) and three translation parameters (T_1, T_2, T_3) of the rigid body transform are computed ($\text{Beta} = [\theta_1, \theta_2, \theta_3, T_1, T_2, T_3]$). The normal matrix (N) and the reference variance ($SO2$) are also computed and utilized to calculate the residual standard errors in rotation ($R\sigma$) and translation ($T\sigma$), where

$$\sigma_{RT} = \text{sqrt}(\text{diag}(SO2 * \text{inv}(N)))$$

$$R\sigma = \sigma_{RT} (1:3)$$

$$T\sigma = \sigma_{RT} (4:6)$$

The rotation matrix is determined from the three rotation parameters ($\theta_1, \theta_2, \theta_3$) as shown in the following equation:

$$R_x R_y R_z = \begin{pmatrix} \cos \theta_2 \cos \theta_3 & -\cos \theta_2 \sin \theta_3 & \sin \theta_2 \\ \cos \theta_1 \sin \theta_3 + \sin \theta_1 \sin \theta_2 \cos \theta_3 & \cos \theta_1 \cos \theta_3 - \sin \theta_1 \sin \theta_2 \sin \theta_3 & -\sin \theta_1 \cos \theta_2 \\ \sin \theta_1 \sin \theta_3 - \cos \theta_1 \sin \theta_2 \cos \theta_3 & \sin \theta_1 \cos \theta_3 + \cos \theta_1 \sin \theta_2 \sin \theta_3 & \cos \theta_1 \cos \theta_2 \end{pmatrix}$$

Due to the formulation of the *coregRotationMatrix* solved by the least squares function, the planar points are rotated around the location of the scanner and not around the origin of FRF coordinates. As a result the rotational components need to be multiplied by the rotational components of the rectification matrix to form the final rotation matrix, and the

translational components are added to the translational components in the rectification matrix to form the final translation vector. Following this procedure, the final rotation matrix is found using the following equation/pseudo code:

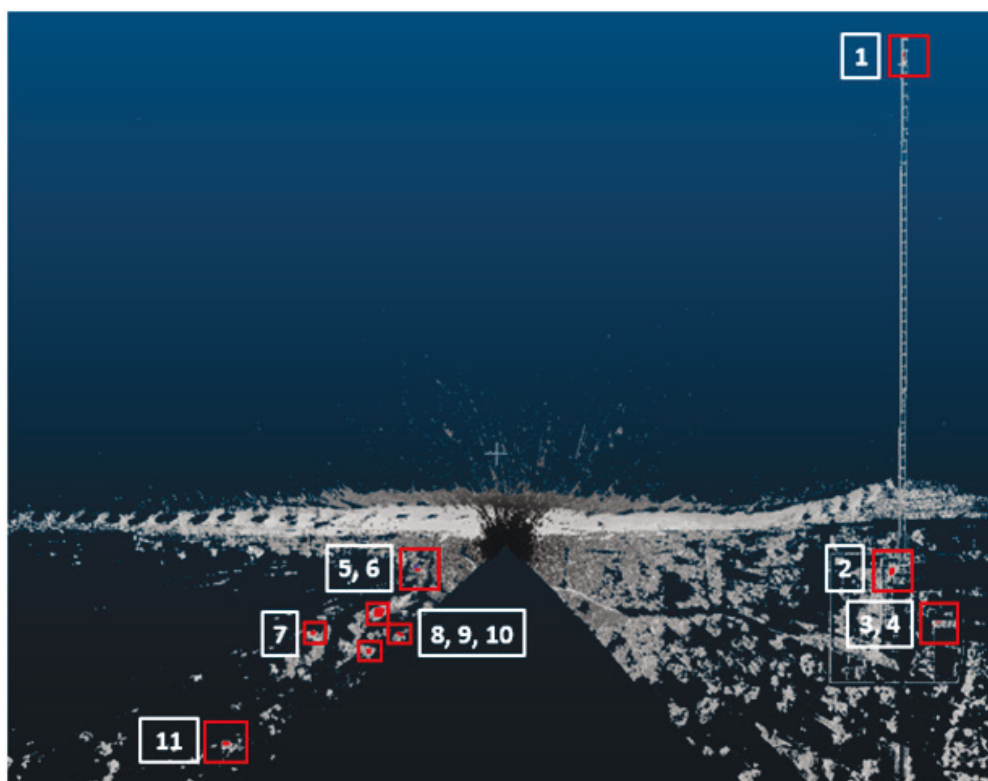
$$rotationMatrix = coregRotationMatrix * rectificationMatrix(1:3, 1:3)$$

The final translation vector is found using the following equation:

$$translationVector = rectificationMatrix(1:3, 4) + [T1; T2; T3]$$

The rotation matrix and translation vector are then combined to form a 4×4 homogeneous transformation matrix that can be used to rotate and translate the raw lidar data from the scanner's own coordinate system to a georeferenced and coregistered coordinate system.

Figure 4-2. Top-down view of a lidar scan (white and gray points) with distance alongshore on the horizontal axis and distance cross-shore along the vertical axis. Red boxes highlight the locations of the 11 planes used in the plane-based coregistration, labeled by number.



The plane-matching approach is initially tried on every scan. If any of the three estimated translational errors generated during the least squares adjustment exceed 0.0025 m, the scan is coregistered again using the location of five permanent reflectors in the scan. As briefly described above, in this coregistration algorithm, the 3D locations of five permanent reflectors near the device are found in the scan and compared to their locations in the baseline scan. The reflectors are found using the *findReflectors.m* function, which trims the data to $1\text{ m} \times 1\text{ m} \times 1\text{ m}$ cubes around the location of the reflector in the baseline scan and then defines the reflector as all points within this cube with a reflectance value in the top 3.5%. The *findReflectors.m* function is located at

DuneLidar/Coregistration/functions/findReflectors.m

The function requires the location of the reflectors in the baseline scan (located in the *parameters* file) and the rectified *xyzr* point cloud as input arguments, and produces a structure that includes the location of the reflectors in the baseline scan, the calculated centroid for each reflector (found using a center of mass on highly reflective returns), the number of points on the reflector, the RMSE for each reflector relative to the baseline location, and the mean RMSE. An unweighted least squares adjustment is then used to determine the rigid transformation (rotation and translation) needed to minimize the difference between the reflector centroid points in the rectified data and reflector centroid points in the baseline scan. The least squares adjustment is done using the *reflectorFit.m* function, located in

DuneLidar/Coregistration/functions/reflectorFit.m

This function requires the reflector centroids from the scan being coregistered as well as the reflector centroids from the baseline scan as input arguments. This function returns the coregistration matrix, the coordinates of the centroids after the least squares adjustment, and the mean RMSE of the reflector centroids. In some cases, the reflector farthest from the device does not have a sufficient number of points to accurately find the location of the centroid. For this reason, the coregistration process is repeated a second time using only the four closest reflectors. Future work will investigate adding high-resolution reflector scans to the current workflow to reduce the number of times this occurs.

4.5.2 Coregistration accuracy assessment

The accuracy of the coregistration is then assessed using the location of three 2D planes in the scan. The first plane is a support beam on the FRF pier. Because of the location and orientation of the pier in FRF coordinates, the beam is seen as a y - z plane with a fixed x -coordinate, which allows for a quantification of the scan offset in the x -direction. The second plane is the front face of the structure immediately to the south of the main FRF building. In FRF coordinates, this building face is an x - z plane with a fixed y -coordinate, which allows for a quantification of the scan offset in the y -direction. The final plane is a portion of the cul-de-sac in the neighborhood to the north of the device. This is an x - y plane with an approximately fixed z -coordinate, which allows for a quantification of the scan offset in the z -direction. Although there is more variability in the z -coordinates of this final plane than in the x - or y -coordinates of the first two planes, this plane was the best option available in the scan. Figure 4-3 shows the location of the planes relative to the lidar, and Figure 4-4 shows each plane individually.

Figure 4-3. Top-down view of a lidar scan (white and gray points) with distance alongshore on the horizontal axis and distance cross-shore along the vertical axis. Red boxes indicate locations of the three planes used in the error analysis.

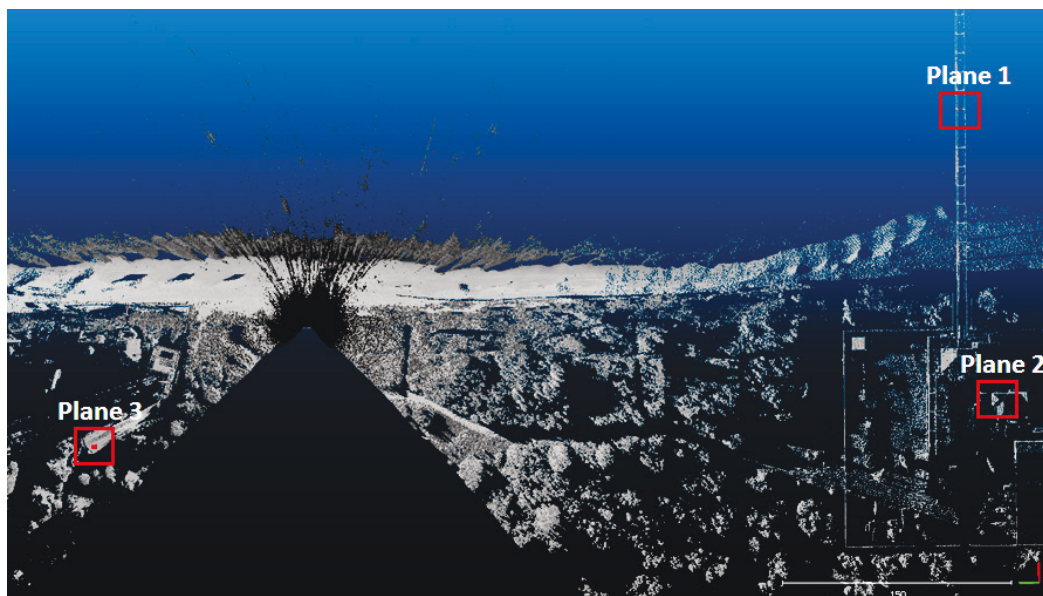
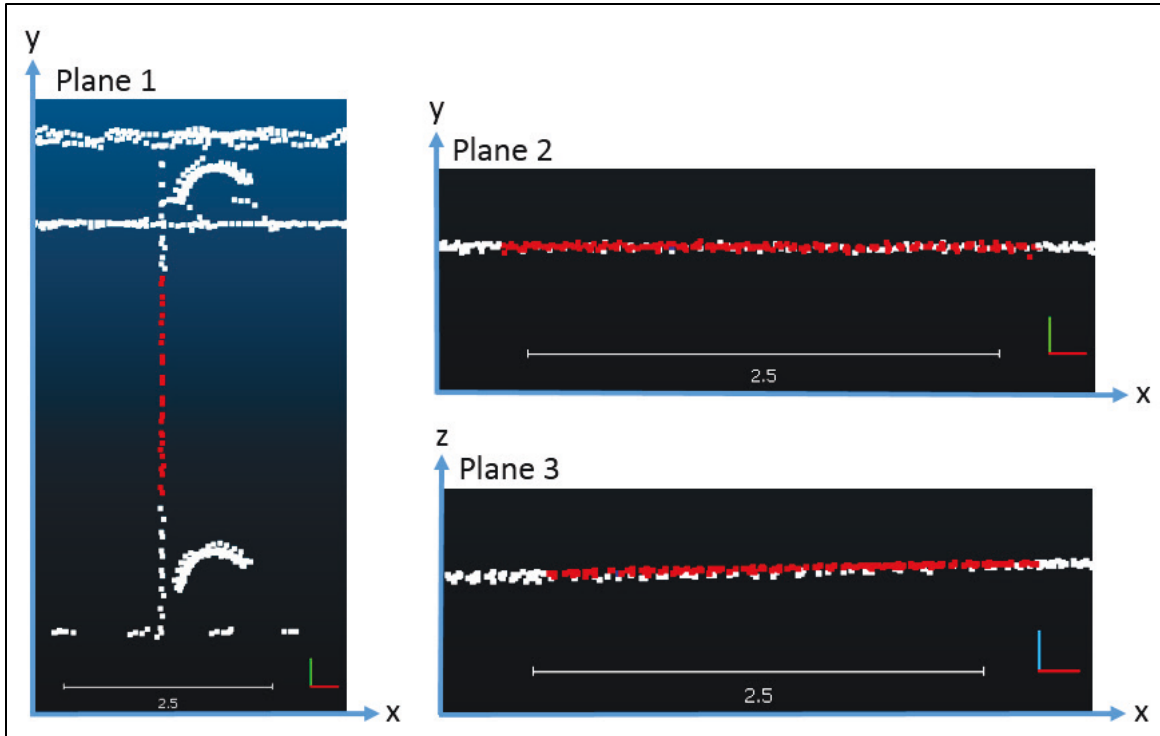


Figure 4-4. The coregistered lidar scan (shown in white) is analyzed for error against Planes 1, 2, and 3, (red). Plane 1 (y - z plane in FRF coordinates) is a support beam on the pier, Plane 2 (x - z plane in FRF coordinates) is the front face of the building to the immediate south of the main FRF building, and Plane 3 (an x - y plane in FRF coordinates) is part of the cul-de-sac in the neighborhood to the north of the lidar.



If the scan was coregistered using only the plane-matching algorithm, the error is determined using these three planes as an additional error metric to the error estimated during the least squares fit. If a second coregistration was conducted using the reflector centroids, these planes are used to select the coregistration matrix that minimizes the error in the x -, y -, and z -direction from the following five cases: (1) the coregistration matrix found using the plane-matching technique, (2) the coregistration matrix found using all five reflectors, (3) the coregistration matrix found using only four reflectors, (4) the coregistration matrix from the previous hour, and (5) the rectification matrix. The error is calculated using the *coregAccuracyAssessment.m* function, located in

DuneLidar/Coregistration/functions/coregAccuracyAssessment.m

The function requires the plane locations and dimensions in the baseline scan (located in the *baseline.mat* file) and the coregistered xyz point cloud as input arguments and produces a structure that includes the centroid of the plane in the baseline scan, the centroid of the plane in the current scan, and the RMSE in each direction relative to the baseline scan.

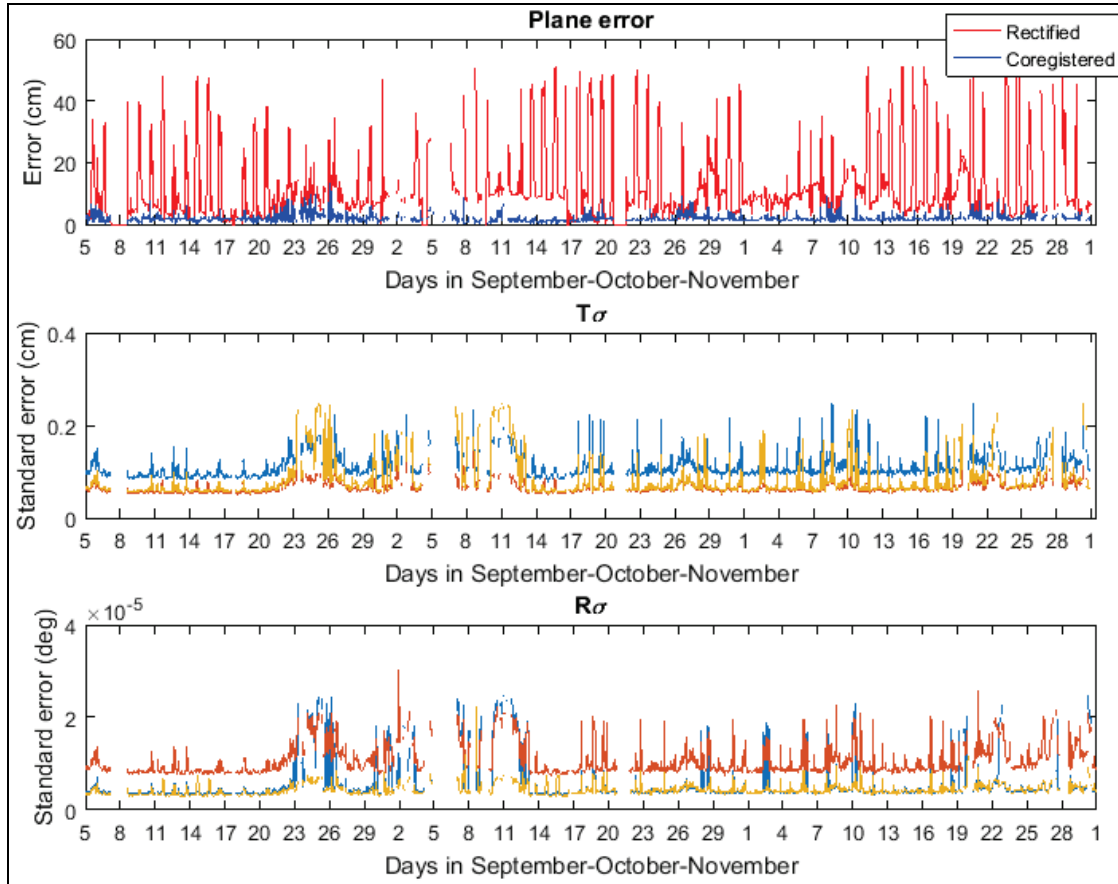
Because the error at each location in the scan is a function of the distance from the scanner, the RMSE in the x -, y -, and z -direction must be normalized for distance before it can be combined. The error is normalized using the *normalizeError.m* function, located in

DuneLidar/Coregistration/functions/normalizeError.m

The function requires the plane locations and dimensions in the baseline scan (located in the *baseline.mat* file) and the coregistered xyz point cloud as input arguments, and produces a structure that includes the centroid of the plane in the baseline scan, the centroid of the plane in the current scan, the RMSE in each direction relative to the baseline scan, and a combined RMSE.

Coregistration errors were assessed using the techniques described above for the period from 5 September to 30 November 2015. A total of 1,792 out of 2,088 scans were coregistered using the plane-matching technique. An additional 202 necessitated coregistration using the reflector centroids. The combined error from the analysis of the coregistration assessment planes for this time period ranged from 0.08 cm to 13.56 cm, with a mean of 1.99 cm and a standard deviation of 1.30 cm. Only 1 out of 1,812 scans had a combined plane error over 10 cm. Figure 4-5 shows a time-series of the errors generated from the plane-matching algorithm ($T\sigma$ and $R\sigma$) and the combined plane error for this time period. Additional error metrics are currently being assessed for potential use, and up to date code can be found on the master branch of *DuneLidar/Coregistration/functions/*.

Figure 4-5. The combined error from the plane error assessment from the rectified scans (in red) and the coregistered scans (blue) (top); the estimated translational error from the least squares adjustment (middle); and the estimated rotational errors from the least squares adjustment (bottom) from 5 September 2015 to 30 November 2015.



4.5.3 Generation of final transformation matrix

If the scan is coregistered using only the plane-matching approach, the final transformation matrix is found using the formulation given in Section 4.5.1.

If a second coregistration using the reflector centroids is required, the matrix that results in the smallest error (from the plane error assessment) is selected from the five cases as the best coregistration matrix for the scan. The coregistration rotation matrix is then combined with the initial rectification matrix to produce a single transformation matrix using the following equation:

$$rotMatrix = coregistrationMatrix * rectificationMatrix$$

With this final transformation matrix, all data acquired during this hour (all three framescans and the linescan) can be rectified in space.

At the end of the coregistration process, the `coreg` structure is saved. This structure includes the final rotation matrix described above; a coregistration “best match,” which indicates which of the five matrices was selected as the coregistration matrix; and an “isCoregistered” flag to indicate that the scan has been coregistered. Additional information, including the rectification matrix, the coregistration matrix, information on reflectors and planes used in the coregistration process and error assessment, the RMSE from the reflectors and from the planes, and information on the baseline scan used are included in the metadata.

4.5.4 QA/QC figure generation

Once the coregistration process is complete, the *zChange* figure, shown in Figure 4-6, is produced and saved using the *makeZDiffPlot.m* function, located in

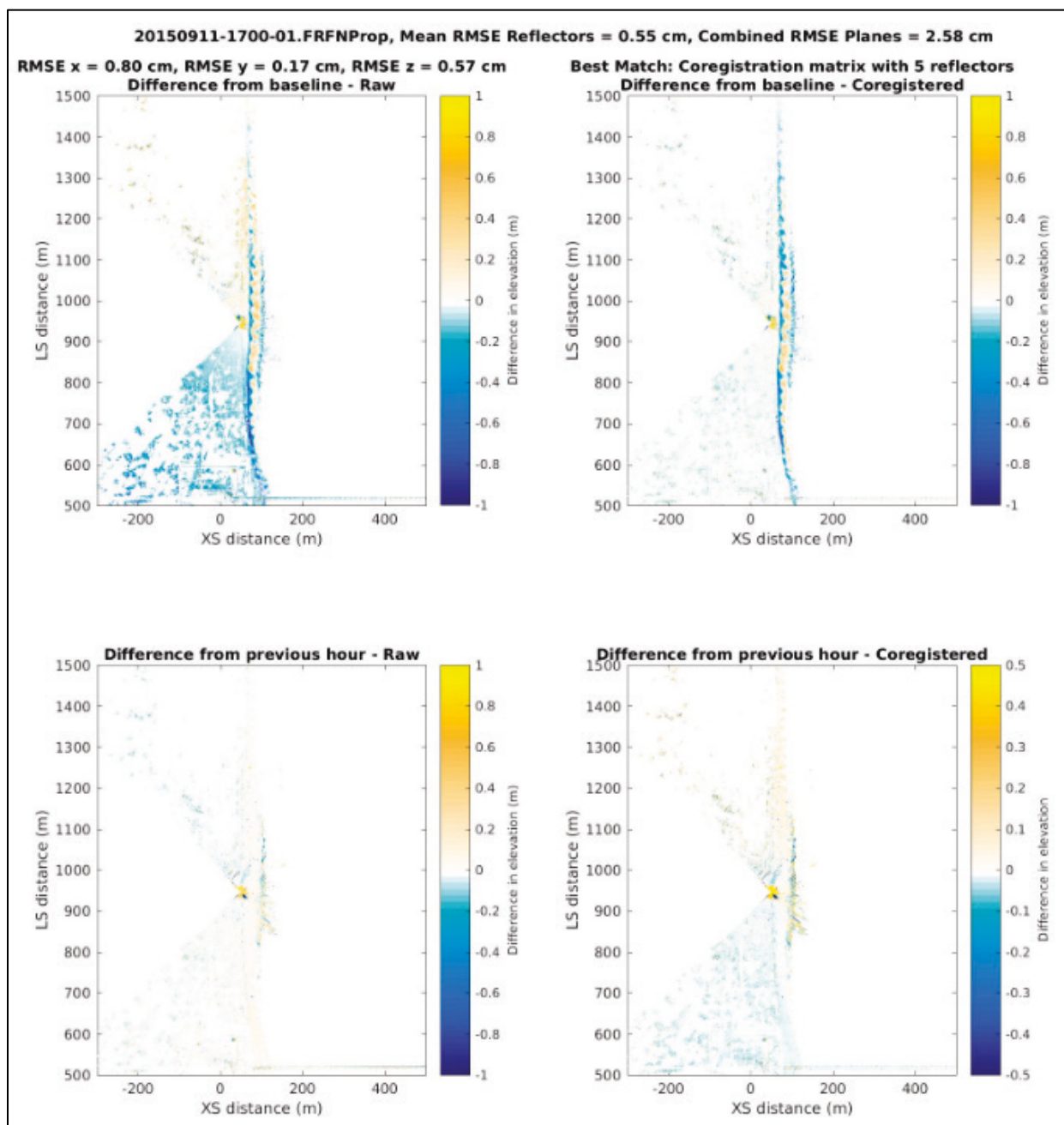
DuneLidar/Coregistration/images/makeZDiffPlot.m

The function requires the *x* and *y* limits of the figure, the raw rectified xyz point cloud from the current scan and from the previous hour, the coregistered xyz point cloud from the current scan and from the previous hour, the baseline xyz point cloud, the desired colormap (located in the *parameters* file), the *rxp* filenames, and the directory where the figure will be saved. The figure includes (1) a plot of the difference between the raw rectified scan and the baseline scan; (2) a plot of the difference between the coregistered scan and the baseline scan; (3) a plot of the difference between the raw rectified scan and the raw rectified scan from the previous hour; and (4) a plot of the difference between the coregistered scan and the coregistered scan from the previous hour. The selected coregistration matrix and error metrics are printed in the figure title. The figure is used to visually assess the quality of the scan coregistration.

An example of the *zChange* figure from 20151008-1200 can be seen in Figure 4-6. In these difference plots, darker colors (blue or yellow) represent areas where there are large differences between the current scan and the baseline scan (top row) or between the current scan and the scan from the previous hour (bottom row), and white represents areas where there are little or no differences or areas where no data are available.

Although large differences are possible offshore and potentially along the beach in well-rectified scans, large differences along the pier or west of the coastline indicate that the scan has not been accurately rectified in space.

Figure 4-6. *zChange* figure example from 20151008-1200 showing (1) a plot of the difference between the raw rectified scan and the baseline scan (top left); (2) a plot of the difference between the coregistered scan and the baseline scan (top right); (3) a plot of the difference between the raw rectified scan and the raw rectified scan from the previous hour (bottom left); and (4) a plot of the difference between the coregistered scan and the coregistered scan from the previous hour (bottom right).



4.6 Framescan processing

Framescan processing is initiated by calling the *ProcessFramescan* function of the script with the same name, located at

DuneLidar/Framescan/ProcessFramescan.m

This function accepts three arguments:

- `dataFolder` – the path to the raw data directory
- `rotationMatrix` – the rotation matrix calculated during coregistration
- `paramsFile` – the path to the parameters file

In its current state, framescan processing includes four steps. First, the data are read, and the three framescans are merged into a single 3D point cloud. Second, noise, water, and objects on the beach are filtered from the point cloud. Third, the filtered points are interpolated onto a regular grid, and a bare earth digital elevation model (DEM) is generated. Fourth, a figure is produced to visually compare the filtered and unfiltered DEM.

4.6.1 Reading and merging data

To more easily work with multiple raw datasets for processing, processing makes use of a helper function called `loadFramescansFromRXP`, which is located at

DuneLidar/Framescan/Functions/loadFramescansFromRXP.m

Because performing a framescan collection consists of three individual scans, it is important to ensure that one extracts the same data fields from each scan before merging and processing them, and `loadFramescansFromRXP` allows for reading consistent data from multiple raw data files in a single function call. This function accepts four arguments:

- `exe` – the path to the *RXPconvert* executable
- `directory` – a directory containing one or more framescan files
- `returns` – a string indicating which lidar returns to read from all files
- `pointvalues` – a cell array of values to retrieve for each lidar return.

Note that except for the data directory, all of these values are defined in the *parameters* file (see Appendix B). At the FRF, the following values are used:

```
params.RXPconvert = '/path/to/RXPconvert';
```

```
params.frame.rxpretains = 'last';
```

```
params.frame.rxpvalues = {'x', 'y', 'z', 'r'};
```

Using these values, read only the last return acquired from each lidar pulse and from each of these returns, followed by the x -, y -, z -coordinates and r , the reflectance value. A complete list of options for both returns and return values can be found in the Appendix C.

The `loadFramescansFromRXP` script will use the *RXPconvert* executable to read this data from every framescan file (with the format **frame*.rxp*) in the provided directory. For each dataset, a 2D array of size $m \times n$ will be created, where m is the number of values requested per return (four in the example above) and n is the total number of returns read from the file. Finally, a structure containing all of the framescan data, where each field is associated with a data file, is returned. The three framescan arrays are then concatenated to produce a single dataset to be filtered.

4.6.2 Filtering and saving LAS files

To generate a bare earth DEM, noise and water points must first be removed. The scans are filtered using the *waterFilter.m* function, which finds the cross-shore location of the waterline at each ϕ value (horizontal coordinate) and then removes all points at that ϕ value with an x -location greater than the determined waterline. For each ϕ value, an approximate water line is defined by finding the location of a large decrease (at least 7 decibels [dB]) in reflectance, which exists because water generally has lower reflectance values than the beach for 1,550 nanometers (nm) light. Once the approximate waterline is defined, the function loops through all ϕ values a second time to refine the waterline by assuming that the waterline cannot have an x - or z -location far from the x - or z -location of the waterline at the adjacent ϕ value. The function filters noise by setting a reflectance threshold of -25 dB and removing all points with a reflectance lower than this threshold value. The *waterFilter.m* function is located in

DuneLidar/Framescan/Functions/waterFilter.m

The function requires the *xyzr* point cloud and the scan rotation matrix (generated during the coregistration process) as input arguments and returns the filtered point cloud and the cross-shore location of the waterline at each phi value.

Using this filtered point cloud, a second filter is used that removes objects from the beach. This filter is included in the *Cloth Simulation Filter* (*csf_filtering*) program, and it separates the point cloud into ground and non-ground points (Zhang et al. 2016). The filter requires the filtered *xyz* point cloud and has six adjustable parameters based on the slope of the terrain, grid size of the filter, vertical threshold distance between ground and non-ground points, and the desired number of iterations.

Once the point cloud has been filtered, the *cloud-filters* program is used to save the data in the LAS format. The *cloud-filters* program has additional features that are currently not being used. The full documentation of the *cloud-filters* can be found in Appendix F. The parameters used to perform the classification are defined in the parameters file. The following options are available:

- `params.frame.filter.intensity` – when saving to the LAS format, each point can have *x*-, *y*-, and *z*-values, as well as an optional intensity value. This flag determines which data to save to the intensity field when creating the LAS file. This value does not need to be the intensity of the return; it is available as an extra dimension of data for each point. The value used is defined by one of the letters given in the `params.frame.rxpvalues` struct. At the FRF, the reflectance value is used, as defined by the letter *r*.
- *(optional)* `params.frame.filter.passthrough` – a boolean value indicating whether or not to run the passthrough filters. The parameters used in the pass-through filters are currently hard coded.
- *(optional)* `params.frame.filter.noise` – the noise filter parameters
- *(optional)* `params.frame.filter.range` – the range filter parameters
- *(optional)* `params.frame.filter.scanthreshold` – the scan threshold value.

The *cloud-filters* program saves both the filtered data and the unfiltered data in LAS format.

4.6.3 Generating bare earth DEMs

Using the filtered and unfiltered LAS files, the bare earth DEMs are made using the *points2grid* program. This program generates a DEM using a local gridding method. The full documentation on the Points2Grid program can be found in Appendix G. The filtered and unfiltered gridded data produced by *points2grid* are saved as *.mean.grid* files, and the filtered gridded data are also saved as a *.mat* file.

4.6.4 Generating a QA/QC figure

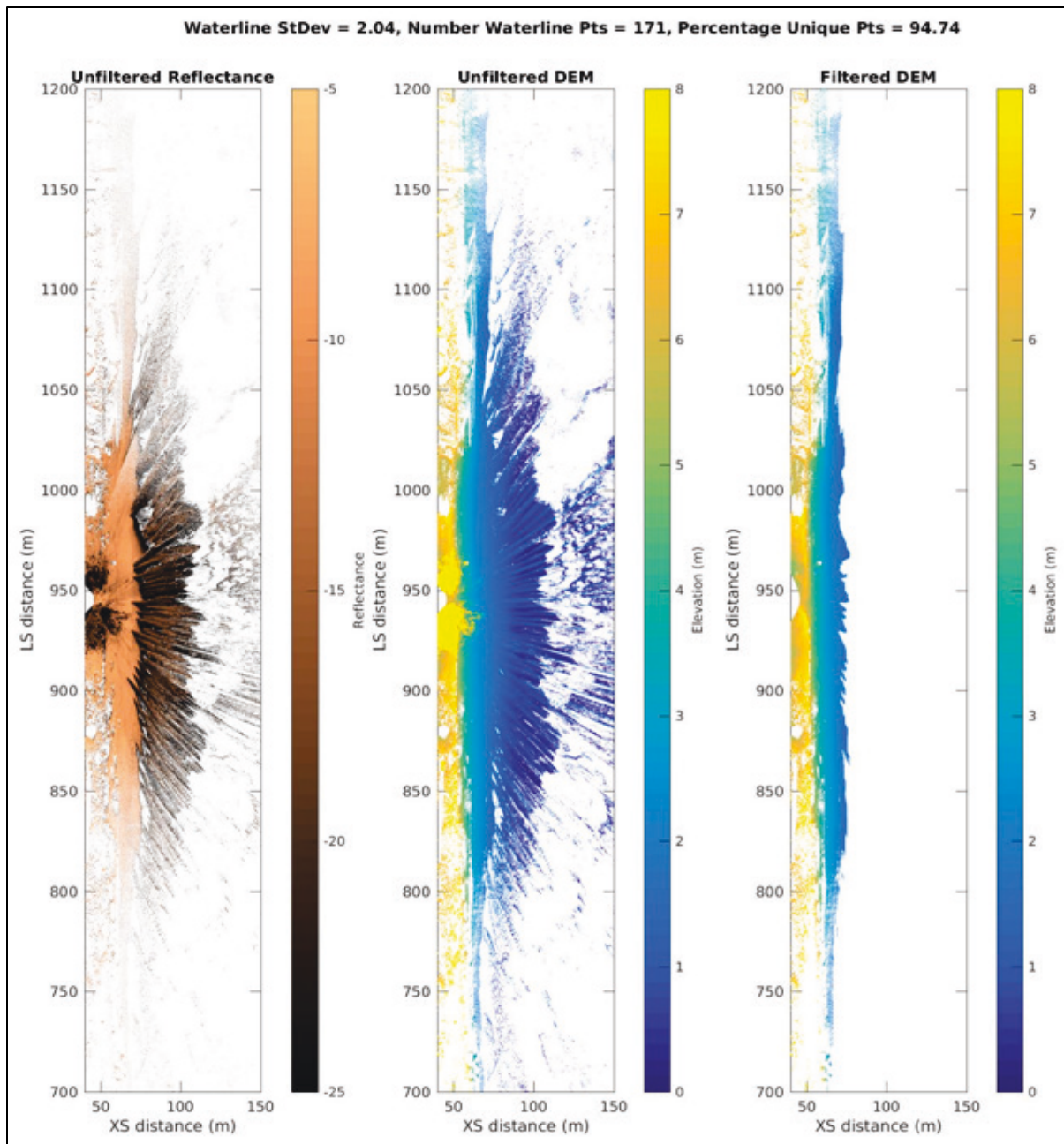
At the end of the filtering process, the DEM figure is produced and saved using the *makeDEMPlot.m* function, located in

DuneLidar/Coregistration/function/makeDEMPlot.m

The DEM figure includes (1) a plot of the unfiltered gridded reflectance, (2) a plot of the unfiltered gridded elevation, and (3) a plot of the filtered gridded elevation. The waterline standard deviation, the number of non-NaN points along the waterline, and the number of unique points along the waterline are printed in the title. The figure is used to visually assess the quality of the water and noise filter. An example of the DEM figure from 20151001-0200 can be seen in Figure 4-7.

Note that because the process is entirely automated, the results of filtering are not perfect. The values used in *waterFilter.m* were chosen specifically for use at the FRF as they generally do a good job at filtering points in this environment and with this instrument setup. Because of the variable nature of the environment being monitored, the filter sometimes fails, requiring QA/QC to be an integral part of the real-time workflow.

Figure 4-7. DEM figure example from 20151001-0200 showing (1) a plot of the unfiltered gridded reflectance, (2) a plot of the unfiltered gridded elevation, and (3) a plot of the filtered gridded elevation with water points removed.



4.7 Linescan processing

Linescan processing is initiated by calling the `ProcessLinescan` function of the script with the same name, located at

DuneLidar/Linescan/ProcessLinescan.m

This function accepts three arguments, along with a number of optional arguments:

- `dataFolder` – the path to the raw data directory
- `rotMatrix` – the rotation matrix calculated during coregistration
- `paramsFile` – the path to the parameters file
- `varargin` – optional parameters that are used for the reprocessing of QA/QC'd data. These parameters are described in Section 4.8.

In its current state, linescan processing of real-time data includes the following steps:

1. Read the raw data and perform pre-processing. Pre-processing involves rectification, basic filtering, and gridding of the data.
2. Generate images based on raw data.
3. Extract the run-up line.
4. Filter data based on run-up line.
5. Perform scientific analysis to generate data products.
6. Save images and data.

Each step will be described in detail in the following sections.

4.7.1 Pre-processing linescan data

4.7.1.1 Rectification

The raw linescan data is read directly from **.rxp* files using the *RXPconvert* program described in Appendix C. A helper script called *makeLineRawRectifiedData* is used to read the data into MATLAB using the *readLidar* script and rectify the data using the rotation matrix generated during coregistration. This script is located at

DuneLidar/Linescan/PreProcessing/rectifyRaw/makeLineRawRectifiedData.m

The script reads only the last return from each lidar pulse, collecting the *x*-, *y*-, *z*-coordinates, the amplitude, and the GPS time for each return. Once data have been read, a 2D matrix is created for each data field, where each column is the value of the data field for a single return and each row of the matrix is a new scanline. For example, the following `Xmat` matrix is laid out in the following manner:

	Return 1	Return 2	Return 3	Return 4	Return 5	
Scanline 1	x1	x2	x3	x4	x5	...
Scanline 2	x1	x2	x3	x4	x5	...
Scanline 3	x1	x2	x3	x4	x5	...

All matrices for an individual collection will be the same size, $m \times n$, where m is the total number of scan lines and n is the maximum number of returns contained in a single scan line (not all scan lines will contain the same number of points). Any scan line that contains fewer than n returns will be padded with not-a-numbers (NaNs). The following matrices are created from the raw data:

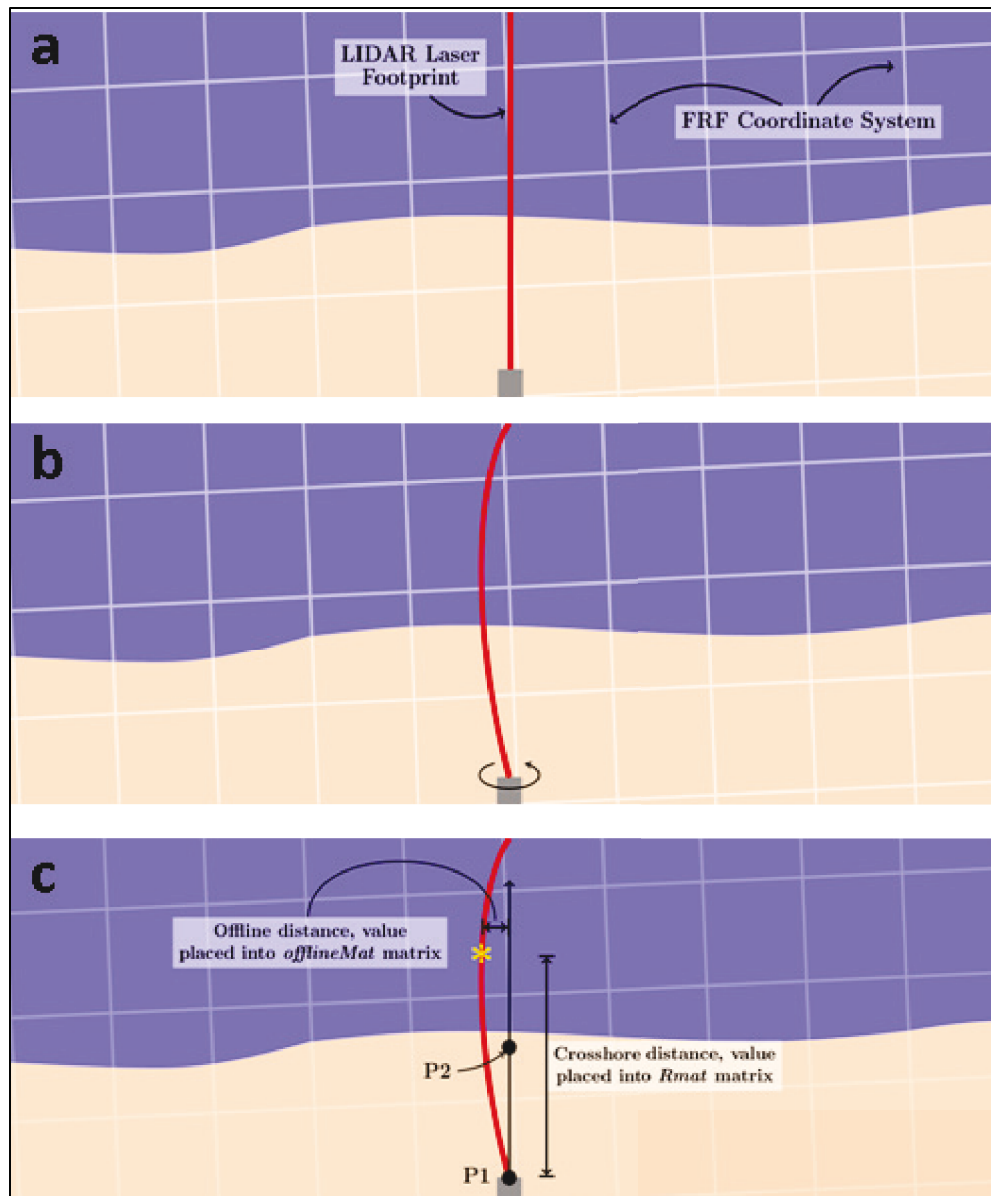
- X_{mat} – x -coordinates in rectified space
- Y_{mat} – y -coordinates in rectified space
- Z_{mat} – z -coordinates in rectified space
- A_{mat} – amplitude of return pulse
- T_{mat} – GPS time at which return pulse was collected.

In addition to these raw data, two additional matrices are calculated:

- R_{mat} – the *downline* distance for each point
- $offlineMat$ – the *offline* distance for each point. The data in both of these matrices are dependent on the values of `params.line.P1` and `params.line.P2` as defined in the parameters file.

These two points are used to define a vector that represents the *downline* direction from the scanner. As shown in Figure 4-8a, the angle of the linescan is not necessarily aligned with the angle of the FRF coordinate system. Any tilt of the linescan plane away from vertical introduces curvature to the linescan on the horizontal plane, as shown in Figure 4-8b. These slight differences are exaggerated in to depict the effect and are normally on the order of -0.2 to 0.2 m in the alongshore direction. The vector created by the points $P1$ and $P2$ provides a straight line, which is considered to be on the cross-shore axis, onto which the linescan data are gridded. The R_{mat} matrix contains the distance of each return along this vector from the point $P1$ (i.e., the cross-shore distance along the vector), and the $offlineMat$ matrix contains the distance of each return away from the vector (i.e., distance from the vector in the alongshore), as shown in Figure 4-8c.

Figure 4-8 (a) The angle of the lidar laser footprint is not necessarily aligned with the FRF coordinate system. (b) Any tilt in the lidar scanner away from vertical will introduce curvature to the lidar laser footprint. (c) The cross-shore and offline distance values of a single return.



Finally, each of the previously described raw data matrices are stored in a MATLAB structure named `lineRawRectifiedData`.

4.7.1.2 Filtering

Once it has been rectified, the raw data are run through a filter that attempts to determine whether each point is noise (people walking on the beach, birds, air particulates, etc.) or a good return (ground or water surface). The filter, defined in

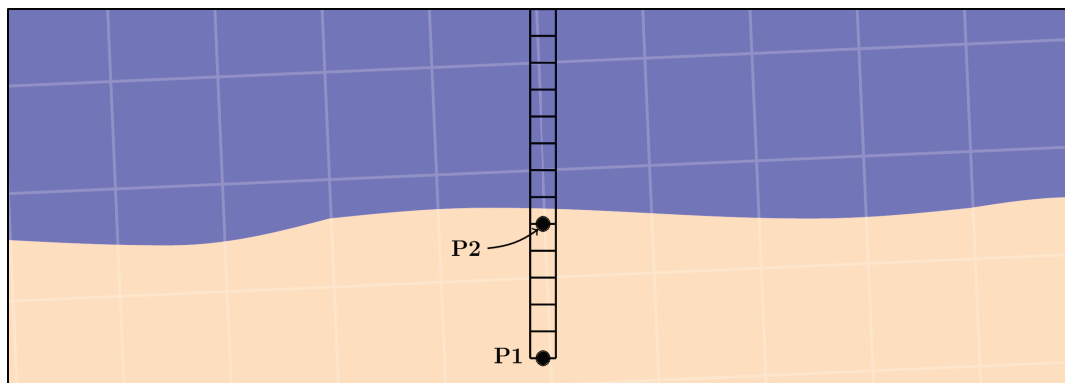
DuneLidar/Linescan/PreProcessing/filterRaw/RangeFilter.m

works on the assumption that if the scanner is doing a low to high scan, the distance of consecutive returns from the scanner should generally be increasing in value. The variables defining how strictly this assumption is enforced are defined in the parameters file. Additional information on how the range filter works can be found in Appendix H. Upon filtering, a MATLAB structure called `lineRawRectifiedNoiseFilter` is created, containing a matrix called `isNoise`, which contains a boolean value indicating whether each point is noise or not (i.e., 1 means the point is noise; 0 mean it is not).

4.7.1.3 Gridding

The raw filtered points are then interpolated onto a 1D cross-shore grid that is aligned with the vector created by points P1 and P2, as shown in Figure 4-9.

Figure 4-9. A sample 1D cross-shore grid that is aligned with the vector created by points P1 and P2.



The grid spacing and extents are determined by the variable `params.line.downline_vector`, defined in the parameters file. At the FRF, the grid extends from -5 to 150 m along the downline vector with a grid spacing of 0.1 m. The gridding is performed in the `makeLineGriddedData` script, located at

DuneLidar/Linescan/PreProcessing/gridRaw/makeLineGriddedData.m

This script is responsible for creating the `lineCoredat` data structure, which contains the following variables:

- `downLineX` – for each grid cell, the mean x -value of every point that falls in the cell
- `downLineY` – for each grid cell, the mean y -value of every point that falls in the cell
- `offlineDist` – for each grid cell, the mean offline distance of every point that falls in the cell
- `downLineXstd` – for each grid cell, the standard deviation of the x -values of every point that falls in the cell
- `downLineYstd` – for each grid cell, the standard deviation of the y -values of every point that falls in the cell
- `offlineDistStd` – for each grid cell, the standard deviation of the offline distance of every point that falls in the cell
- `downlineDistStd` – for each grid cell, the standard deviation of the downline distance of every point that falls in the cell
- `lidarLocation` – the lidar’s center in rectified coordinates

The variables created in the `lineCoredat` structure are then used by the script to create the `lineGriddedData` structure, which contains the following variables:

- `zGrid` – an $m \times n$ matrix of z -values. Cells that contain no points are filled with NaN.
- `aGrid` – an $m \times n$ matrix of amplitude values. Cells that contain no points are filled with NaN.
- `zGridInterp` – an $m \times n$ matrix of z -value with small gaps of NaN cells filled with linearly interpolated values. A maximum of 10 consecutive NaN cells will be filled with interpolated values.
- `aGridInterp` – an $m \times n$ matrix of amplitude values with small gaps of NaN cells filled with linearly interpolated values. The gaps that are filled match the gaps that were filled when creating the `zGridInterp` variable.
- `tGPSVector` – a $1 \times m$ vector containing the average GPS time for each scan line
- `tGPSVectorRange` – a structure containing the min and max GPS time for each scan number
- `Hz` – the average full scan line record frequency in hertz

For each of the matrices included in the `lineGriddedData` structure, the matrix size will be $m \times n$ where m is the number of scan lines, and n is the number of cells in the 1D grid. The gridding of the data itself is performed

by the `interp1NaNThresh` function, which is located in the Utilities repository at:

Utilities/Gridding/interp1NaNThresh.m

interp1NaNThresh.m is a simple interpolation algorithm that linearly interpolates between data points in the cross-shore but only through gaps that are a maximum distance wide and/or tall (as defined by the inputs), to generate evenly spaced data points at a specified resolution. The inputs to *interp1NaNThresh.m* are the original down-line distances of the lidar points (`Rmat`), the elevations of the lidar points (e.g., `Zmat`), the gridded downLineDistance vector (`params.line.downline_vector`), the maximum cross-shore distance to interpolate through (10 m), and the maximum elevation jump to interpolate through (5 m). Interpolation is needed to create a continuous sea-surface from which hydrodynamic statistics can be calculated for cases when the backs of waves are shadowed or there is no foam on the water surface.

4.7.2 Generating raw data images

Before any additional processing occurs, a few images are generated from the raw data. These images are useful in the case that the processing crashes in the subsequent steps before any new data or images can be saved, as they may shed some light as to why the processing was unsuccessful. Currently, only two images are created. The first, called a *raw point plot*, shows a cross-shore profile of all returns in the linescan data, as shown in Figure 4-10. This image confirms that raw data were collected but includes noisy returns off of scatterers in the air that have not yet been filtered out. The second, called an *instant linescan plot*, shows the cross-shore profile of a set number of scan lines, each with their own color scheme, as shown in Figure 4-11. This image is used to qualitatively assess how many returns the lidar is receiving from the water surface for the current environmental conditions, as well as an initial qualitative look at wave heights and shapes in the inner-surf zone. Data gaps in the cross-shore profile of the sea-surface result from either shadowing on the back face of waves or from still, un-foamy water surfaces (Brodie et al. 2015). The plotting options for both of these plots can be set in the *parameters* file, as described in Appendix B.

Figure 4-10. A cross-shore profile of all points in a raw dataset.

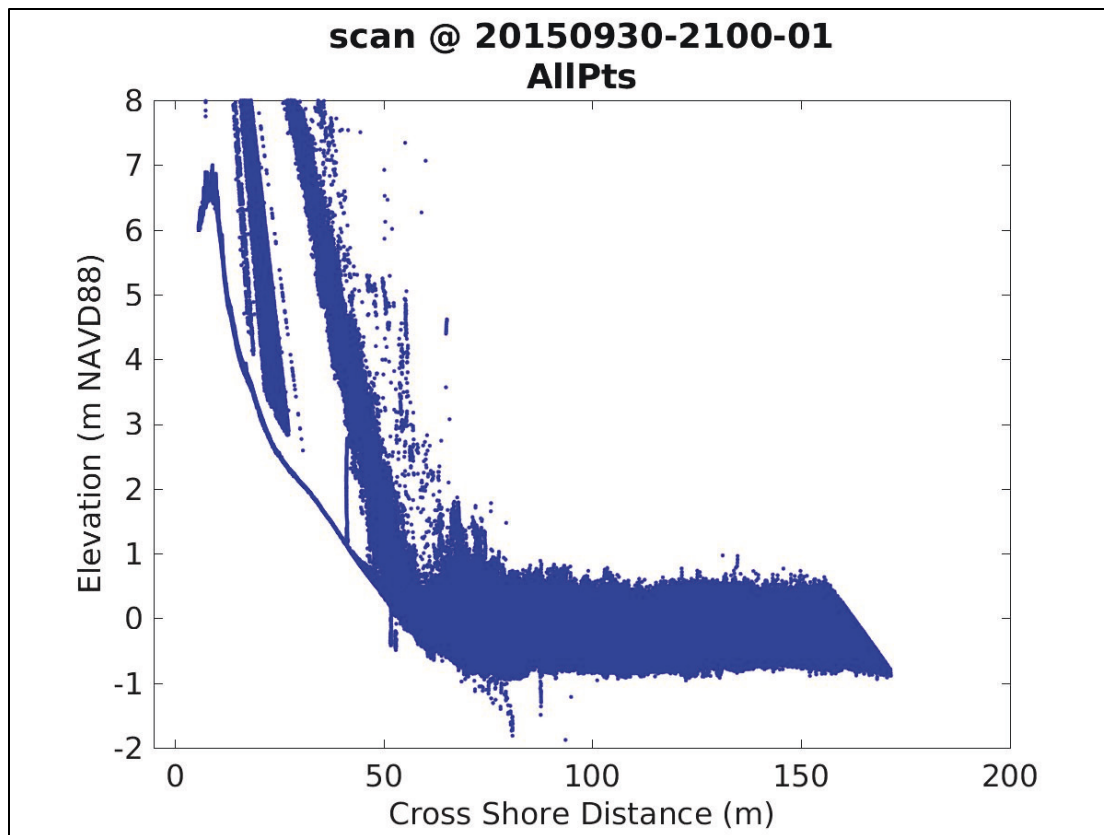
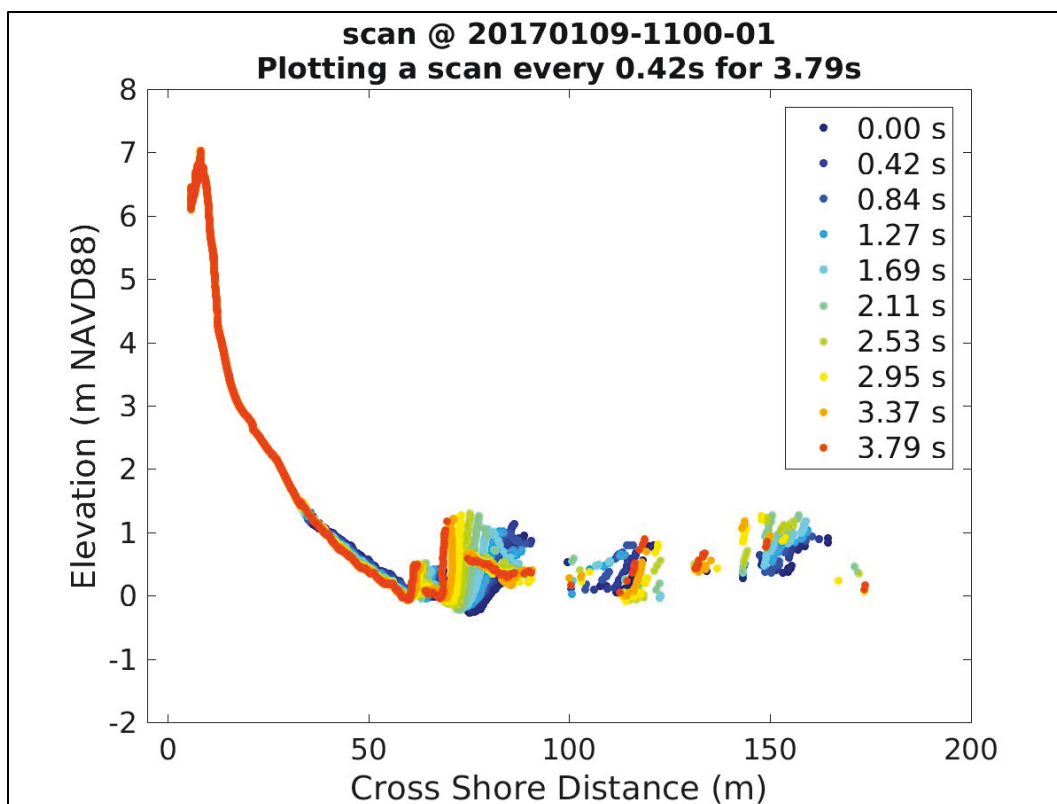


Figure 4-11. A cross-shore profile of 10 individual scan lines from a filtered dataset.



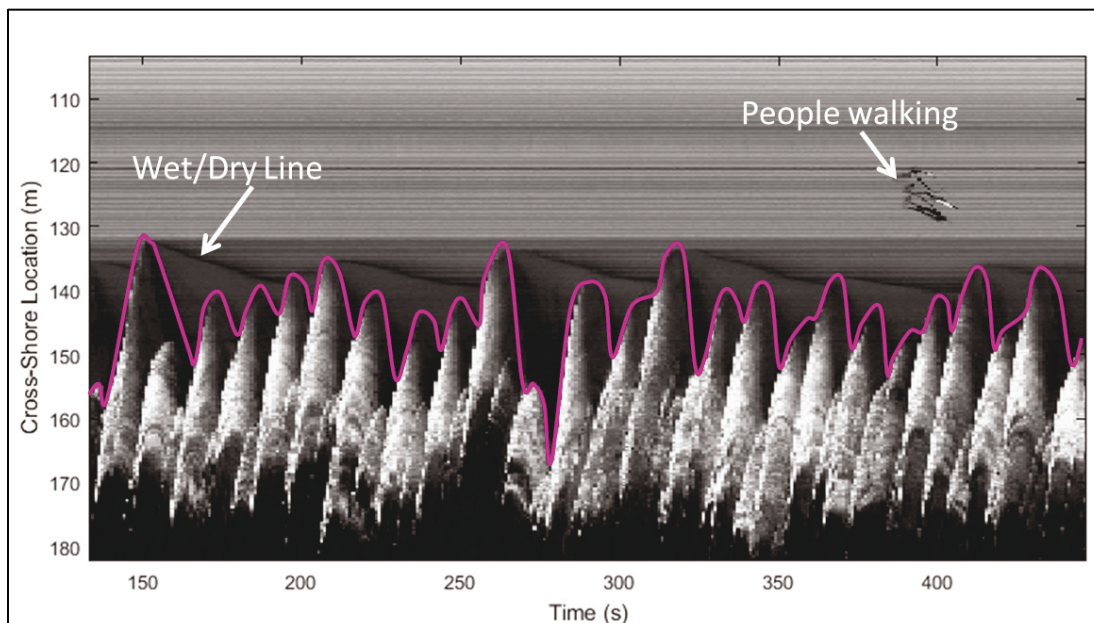
4.7.3 Runup line extraction

Once the raw data has been filtered, the next step is to determine the location of the runup line. The runup line can be defined as the location of the beach-water interface at any point in time. Here, the single cross-shore profile limits the beach-water interface to a single point. Therefore, when considering the cross-shore beach profile the runup line is defined as the evolution of the cross-shore location of the beach-water interface through time. To visualize this distinction, refer to Figure 4-12 and Figure 4-13. Figure 4-12 shows the runup line for a large swath of beach at a single point in time. On the other hand, Figure 4-13 shows the runup line for a single cross-shore profile as it evolves through time.

Figure 4-12. The cross-shore position of wave runup (magenta) along the beach at a single point in time from Argus video imagery. The blue line represents a single cross-shore profile through time.



Figure 4-13. The cross-shore position of wave runup (magenta) for a single cross-shore profile through time (e.g., along the blue line in Figure 4-12). These data are from video imagery for ease of viewing but are similar to the data captured by the lidar scanner.



Extracting the location of the runup line from the gridded data is done using the `calculateRunupLine` function located at:

DuneLidar/Linescan/PreProcessing/calcRunupLine/calculateRunupLine.m

This function calls `getRunupIndex` function, which attempts to determine the cross-shore location of the instantaneous runup line during each linescan by looking at elevation changes through time at each cross-shore location in the linescan. Sand-level changes on the lower foreshore occur on the time scale of infragravity and swell-sea waves (Howd and Holman 1987), and so the lowest elevation at any given point in the swash cannot simply be assumed to always represent the elevation of beach topography during the linescan. To account for this, data at each cross-shore location are filtered using a running minimum over time windows ranging from 1.2 to 30.0 seconds to generate multiple temporally varying minimum surfaces. Each minimum surface is then subtracted from the original line-scan elevation data, and a threshold of 0.015 m is used to threshold the resulting difference maps for all minimum surfaces. An exceedance threshold can then be used to identify the most shoreward location in each linescan with an elevation > 0.015 m above the average minimum surface to define the run-up–beach intersection (the maximum run-up location). Points seaward of this location are classified as water, and points landward are classified as the foreshore. The run-up extraction algorithm correctly separates water and land points approximately 85% of the time. Incorrect digitization is normally caused by relict foam or seaweed which often interferes with the digitizing. Therefore, the present methodology requires each collection to be manually QA/QC'd to ensure proper separation of water and land points (described in Section 5, below). Future work will aim to improve this portion of the code to reduce the reliance on hand-checking for data quality. The `calculateRunupLine` script creates and returns the `lineRunupCalc` data structure, which contains the following variables:

- `zDiffCumulative` – Cumulative count of the number of times the data at a cross-shore location was classified as water above various running-minimum filters.
- `downLineIndex` – An $m \times 1$ array of the down-line index of the runup position for each scan line
- `downLinePosition` – An $m \times 1$ array of the down-line distance of the runup position for each scan line
- `Z` – An $m \times 1$ array of the z -value at the runup position for each scan line
- `isQAQC` – A numeric flag indicating whether the data has been quality controlled. Automatically set to zero for real-time data.
- `QAQCmeta` – A data structure containing metadata about QA/QC

- `rawRectifiedDataIsWater` – An $m \times n$ masking matrix indicating whether each point in the `rawRectifiedData` matrices are water (i.e., 1 means the point is water; 0 means it is not). In this matrix, n is equal to the maximum number of returns in any scan line (i.e., it will have the same dimensions as the `rawRectifiedData` matrices).
- `griddedDataIsWater` – An $m \times n$ masking matrix indicating whether each point in the `lineGriddedData` matrices are water (i.e., 1 means the point is grid cell is water; 0 means it is not).

Unless otherwise indicated, for each of the vectors and matrices in the `lineRunupCalc` data structure, m is equal to the number of scan lines and n is equal to the number of grid cells.

4.7.4 Filtering

Once data have been classified as water or land, tighter filtering can be performed on the topography to remove any final outliers that were not identified during the initial filtering by using the assumption that any high-frequency oscillations in the topography at a given cross-shore location are outliers or noise (not swash). To accomplish this, the `filterForeshore` script is used, which is located at

DuneLidar/Linescan/PreProcessing/calcRunupLine/filterForeshore.m

This script reads in the foreshore data grid, the `isWater` Boolean matrix, and the linescan frequency in hertz, and temporally filters the data using a lowpass filter to remove outliers. The filter defines topographic changes of more than 0.25 m in 30 seconds and 0.05 m in 5 seconds as noise. The removed data points are replaced by interpolating linearly in time and space to generate a new value. The output is a matrix of filtered foreshore elevations that is the same size as the input.

4.7.5 Finalizing the pre-processing

The final steps in preparing the data for analysis are to create the `fGrid` variable and save all pre-processed data to file. The `fGrid` variable is used in the quality control workflow to help produce a more accurate runup line. It is created at this point in the processing because it is calculated using the filtered data as well as the runup line. The following steps are used to generate the `fGrid` variable:

1. Extract the foreshore from the `zGrid` variable using the runup line (i.e., remove all water from the `zGrid` variable).
2. Fill gaps in the foreshore variable by interpolating through time.
3. Create the `fGrid` variable by subtracting the interpolated foreshore variable from the `zGrid` variable.

The `fGrid` variable contains the wave runup data with the underlying foreshore subtracted out. When a wave breaks and runs up the shore, the lidar scan only captures the elevation of the water surface not the elevation of the ground surface underneath the wave runup. However, the ground surface elevation underneath the wave runup can be inferred by using ground surface measurements taken before and after the wave has run up the beach by interpolating the foreshore elevation through time. The `fGrid` variable amplifies regions with changing elevation, which normally depicts the location of the runup line. The variable therefore becomes very useful in iteratively refining the quality of the digitized runup line, as described in Section 5.

Finally, after the `fGrid` variable has been calculated, the `lineCoredat`, `lineGriddedData`, `lineGriddedFilteredData`, `lineRunupCalc`, `lineRawRectifiedNoiseFilter`, `lineRawRectifiedData`, and `fGrid` variables are all saved to a single `*.line.data.mat` file. The pre-processing is fairly computationally intensive, so data are saved at this point to allow for easier debugging as well as easier reprocessing should an error occur during any of the subsequent processing steps.

4.7.6 Scientific analysis and data product generation

Once pre-processing is complete, data products summarizing waves, water levels, and morphology changes are generated by performing scientific calculations on various portions of the data. Products derived from the runup line itself, such as minimum and maximum wave runup, are calculated in the `DoRunupScience` script, located at

DuneLidar/Linescan/Runup/DoRunupScience.m

Hydrodynamics data products, which are generated using only water data and include things such as minimum/maximum water levels and significant wave heights, are calculated in the `DoHydrodynamicsScience` script, located at

DuneLidar/Linescan/Hydrodynamics/DoHydrodynamicsScience.m

Virtual wave gauge data products are similar to the hydrodynamics data products, with the exception that they are calculated at discrete locations in the cross-shore. These products also include data such as minimum/maximum water levels and significant wave heights. Wave gauge data products are calculated in the `DoWaveGaugeScience` script, located at

DuneLidar/Linescan/WaveGauges/DoWaveGaugeScience.m

Finally, morphological data products, which describe the topography of the beach are calculated using only foreshore data, or data that fall on the landward side of the runup line. These products include data such as beach curvature and slope, and are calculated in the `DoMorphologyScience` script, located at

DuneLidar/Linescan/Runup/DoMorphologyScience.m

An in-depth description of all data products generated by each of the previously described scripts can be found in Section 5.

4.7.7 Saving images and data

The final step in linescan processing is to generate two images that can be used for both informative and debugging purposes. In concert with the images generated during pre-processing, these plots and charts provide a simple way to explore the quality of the raw data, the accuracy of the runup line, and the validity of the data products that have been generated. The plots shown in Figure 4-14 and Figure 4.15 are generated during this post-processing step.

Figure 4-14. A plot showing samples of raw data as well as morphological, hydrodynamics, and runup line data products.

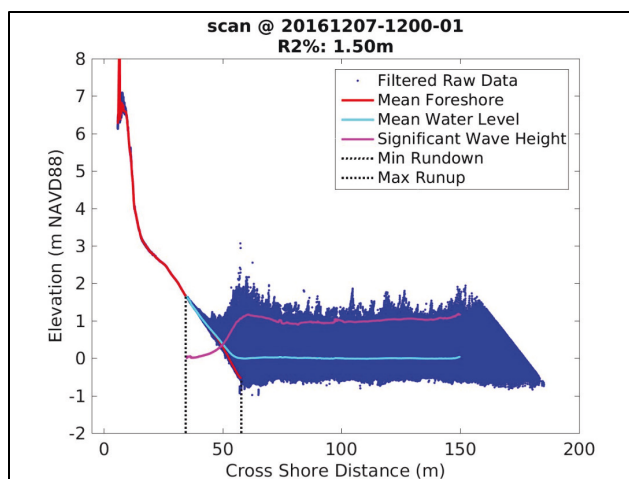
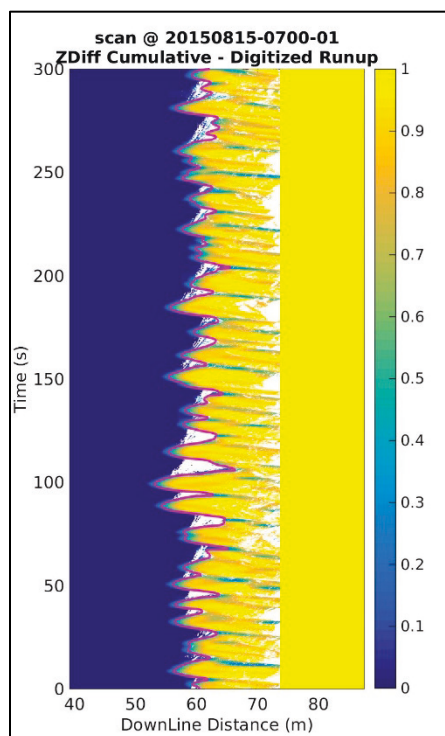


Figure 4-15. A plot showing the *zDiffCumulative* variable values as well as the runup line in terms of its distance down-line through time.



4.8 Reprocessing data

As Section 5 describes, after real-time data have been collected, it is passed through a quality assurance and QA/QC workflow that involves modifications to the runup line by a trained user. When the QA/QC has

finished, the data must be reprocessed in order to re-generate data products based on the updated runup line. This reprocessing is performed by the same scripts that performed the initial real-time processing, with a few minor exceptions and modifications.

First, the *MasterProcess.m* script, which is responsible for deciding which dataset to pass through the processing workflow, detects a dataset eligible for reprocessing. It will then call on the *ReprocessDirectory.m* script to perform the reprocessing. This script is identical to the *ProcessDirectory.m* script, with the addition of the following four parameters:

- `type` – a string describing the type of QA/QC that has been performed on the data. Currently, only “runup” is supported.
- `monthlyDir` – the path to a directory containing monthly netCDF data files, described in Section 5.
- `varargin` – a variable number of arguments, based on the value of `type`. For a type of “runup”, the following must be included:
 - `'runupFile'` - The path to the file containing the new runup line (as created by RunupTool). This will be the `*.line.data.mat` file.
 - `'runupVariable'` - The variable in the file that contains the new runup line.

Note that the two variables `'runupFile'` and `'runupVariable'` are both included in the reprocessing queue file that is maintained by *RunupTool*, as described in Section Setup.

The *ReprocessDirectory.m* script, after validating all data required for reprocessing, calls the *ProcessLinescan.m* script to begin the reprocessing. The *ProcessLinescan.m* script, which is used to perform both real-time and reprocessing of data, also contains the `varargin` parameter and will determine which processing steps to complete based on the arguments included. If none are included, the normal real-time data processing is performed. If the `'runupFile'` and `'runupVar'` variables are included, reprocessing will be performed. If reprocessing and the `'monthlyDir'` variable are included, the reprocessed data products will be added to the appropriate netCDF file in the monthly data directory, as described in Section 6.

When *ProcessLinescan.m* performs the reprocessing of quality-controlled data, the following two steps differ from those of the real-time data processing:

- **Preprocessing**, as described in Section 4.7.1, **is skipped**. These matrices generated during this step, which includes rectified and gridded data, are identical for both real-time and reprocessing, so the data are loaded from the **line.data.mat* file.
- **The lineRunupCalc struct is recalculated**. The *calculateRunupLine.m* can optionally receive a QA/QC'd runup line. When it does, it skips runup line extraction and recalculates the `lineRunupCalc` variables based on the provided runup line.

Once the `lineRunupCalc` struct has been recalculated using the QA/QC'd runup line, the rest of the reprocessing workflow is identical to the real-time processing workflow. As such, all data products will be recalculated using the QA/QC'd runup line.

5 Quality Assurance and Quality Control (QA/QC)

The data products generated by the real-time data processing workflow described in Section 4 are subject to errors and inaccuracies due to the fact that the data used to create them are automatically collected from an active natural environment. Once the real-time data processing has been completed, a level of quality control is required to verify the accuracy of the data products. Currently the FRF employs two quality-assurance tools. The first is a near-real-time monitor of the scanner itself called *Lidar Monitor*, which is useful in determining which datasets may contain corrupted data as a result of a hardware error, and the second is a program called *RunupTool*, which is used to validate the accuracy of the runup line used in the real-time processing workflow.

5.1 Lidar Monitor

Lidar Monitor is a web page, located at <http://www.frf.usace.army.mil/lidar> that provides basic information about that status of the lidar scanner. As part of the data collection process, the *StationaryTowerCollect.sh* script transfers the scanner status and scanner error files that are generated by the *LidarCollect* program to the FRF's public-facing web server. The *Lidar Monitor* web page then provides a convenient display of the data contained in these files, as shown in Figure 5-1.

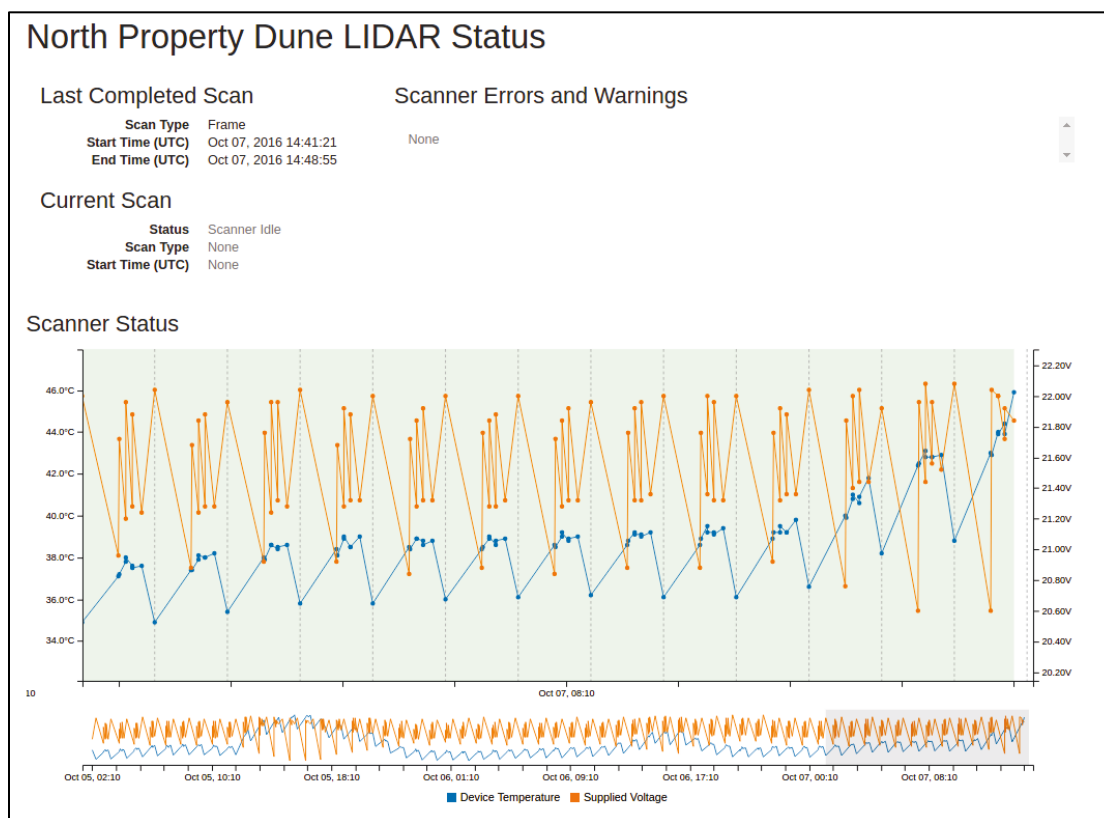
The web page displays the following data:

- the type, start time, and end time of the last scan performed by the lidar scanner
- the current status of the scanner, either idle or scanning
- if the scanner is currently scanning, the type and start time of the scan currently being performed
- a list of error messages and warnings returned from the scanner any time over the past two and a half days
- an interactive plot of the past two and a half days of device temperature and supplied voltage data.

This web page is meant to provide enough information that an FRF employee can easily check the current status of the lidar scanner from any location to ensure that data is being collected. Error messages and spikes

in the scanner temperature/voltage may also be used as a first pass diagnostic tool for locating potential errors in the raw data.

Figure 5-1. The *Lidar Monitor* web page.



5.2 RunupTool

RunupTool is a program that has been developed to aid in assuring the quality of the runup line used to perform scientific calculations on lidar collected beach morphology and wave hydrodynamics data. The algorithmically determined runup line (derived from linescan data) is prone to error in various circumstances. Errors in the location of the runup line result in incorrect water points being included in beach morphology data, and/or beach morphology data being included in the water points.

After completion of automated data processing, a trained user must review the quality of the runup line to validate the quality of the data products that have been produced. *RunupTool* allows the user to both visually inspect the quality of the runup line as well as make corrections to errors in the runup line and introduce the quality-controlled data into the reprocessing workflow. *RunupTool* has been developed to work

specifically with MATLAB data files that have been produced by the real-time data processing workflow outlined in this report. The following sections describe how to set up and use *RunupTool* as part of the automated data processing workflow.

5.2.1 Setup

The first time *RunupTool* is opened, four file/directory locations need to be set. The user will only be prompted to set these values the first time opening *RunupTool*, but the values can be changed through the Settings menu. There are three file locations and one directory location that need to be set, outlined as follows:

- **Status file** – must have a *.ini* extension. The status file keeps track of the state of every dataset that has been opened with *RunupTool*. This allows a user to keep track of which datasets have been modified by *RunupTool* between sessions. In an environment in which multiple users are working on the same datasets on a single network, this file is used to coordinate between *RunupTool* sessions, preventing multiple users from editing the same data. A single status file should be located on the server and used by all *RunupTool* instances in this environment.
- **Users file** – must have a *.ini* extension. The users file keeps track of registered *RunupTool* users. When working with data, edits and commits must be attributed to a user, as outlined in the following Users section. This approach allows for user data to be saved between *RunupTool* sessions so that user profile information only needs to be set once. This file is used to coordinate between *RunupTool* sessions and dataset use by different parties on the same network. By placing this file on the network and using it for all *RunupTool* instances, a user can work on any machine on the network without having to set their user information on each computer.
- **Temporary file directory** – this directory will contain all temporary data associated with *RunupTool*. This includes edits to the runup line and commit status. Each dataset edited with *RunupTool* will have an associated temporary file located in this directory. Placing this directory on the network allows for access to edited runup lines from any machine also connected to the network.
- **Reprocessing queue** – this file, which is a text file, is used by the real-time data processing workflow to initiate the reprocessing of datasets with updated runup lines. When a user is finished editing the runup line, information about the dataset will be appended to the end

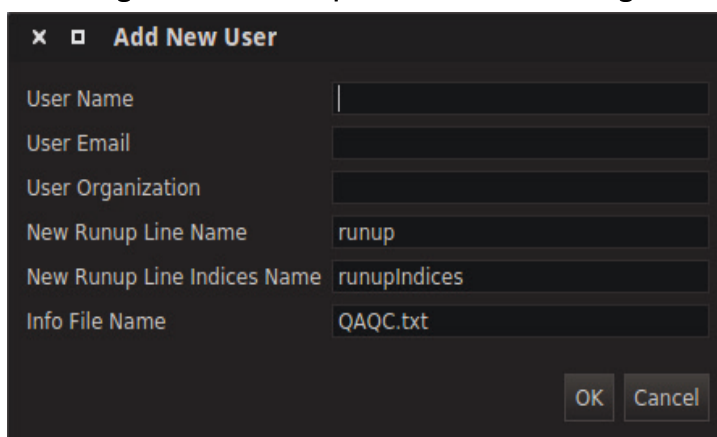
5.2.3 Users

When making changes to the runup line with *RunupTool*, all edits must be attributed to a user. *RunupTool* allows a user to set up a simple profile, which will be used to include attribution data with all datasets that the user edits. Each time *RunupTool* is opened, the user must either select their user profile from the User Dropdown or create a new user profile. A user profile can be created by opening the “User Dropdown” and selecting the “Add User” option. A dialog window, as shown in Figure 5-3, with the following options will open:

- **Name** – the user’s name
- **Email** – the user’s email address
- **Organization** – the user’s organization
- **Runup name** – the variable name to give the edited runup line
- **Runup indices name** – the variable name to give the binary array indicating which indices in the runup line have been edited

It is recommended that the default values for the runup name and runup indices name are used. Once all values have been set, click “OK” to add and set the new user as the current user.

Figure 5-3. The RunupTool Add New User dialog.



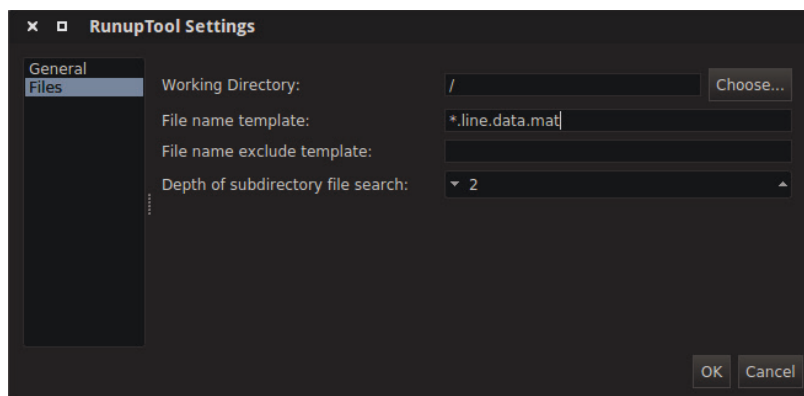
Field	Value
User Name	
User Email	
User Organization	
New Runup Line Name	runup
New Runup Line Indices Name	runupIndices
Info File Name	QAQC.txt

5.2.4 Discovery

Once the program has been properly initialized, and a user has been selected, the user must set up data discovery. Data discovery allows the user to more quickly move between datasets by only displaying relevant files from datasets. Data discovery settings are accessed through the

settings menu by clicking on the settings cog at the top of the file viewer. This will bring up the settings menu, shown in Figure 5-4.

Figure 5-4. The RunupTool settings menu.



The settings in this menu will determine which files are displayed in the file viewer. The settings are as follows:

- **Working Directory:** The top-level directory in which data discovery will begin. Only files located within or below this directory will be available for display in the file viewer.
- **File name template:** The regular expression defining the names of files that will be displayed in the file viewer. Note that only the filename is used in the regular expression, the full directory path is not included. In the example shown, any file ending in *.line.data.mat* will be included in the file viewer.
- **File name exclude template:** A regular expression defining file names to exclude from the file viewer. This allows the user to further narrow the list of files displayed in the file viewer by excluding certain files that have passed the file name template.
- **Depth of subdirectory file search:** The number of directory levels below the working directory to traverse when looking for files. For example, 2 will search all directories contained in the working directory as well as all directories contained by those directories. A value of 0 will only include files directly in the working directory.

Once these values have been set, press the “OK” button, and the working directory will be displayed above the file viewer, and file viewer will populate.

5.2.5 Visualizing data

Files in the file viewer can be opened by double-clicking. If a file was open and there were data plotted, the existing plots will be repopulated with the data from the newly opened file. When a file is opened in *RunupTool*, the variables from that file, along with their row and column dimensions, will be displayed in the variable display. Any variable that has more than one row and one column can be plotted by double-clicking on the variable. Upon double-clicking a variable, the dialog shown in Figure 5-5 will provide plotting options.

Figure 5-5. RunupTool plotting options.



The options are as follows:

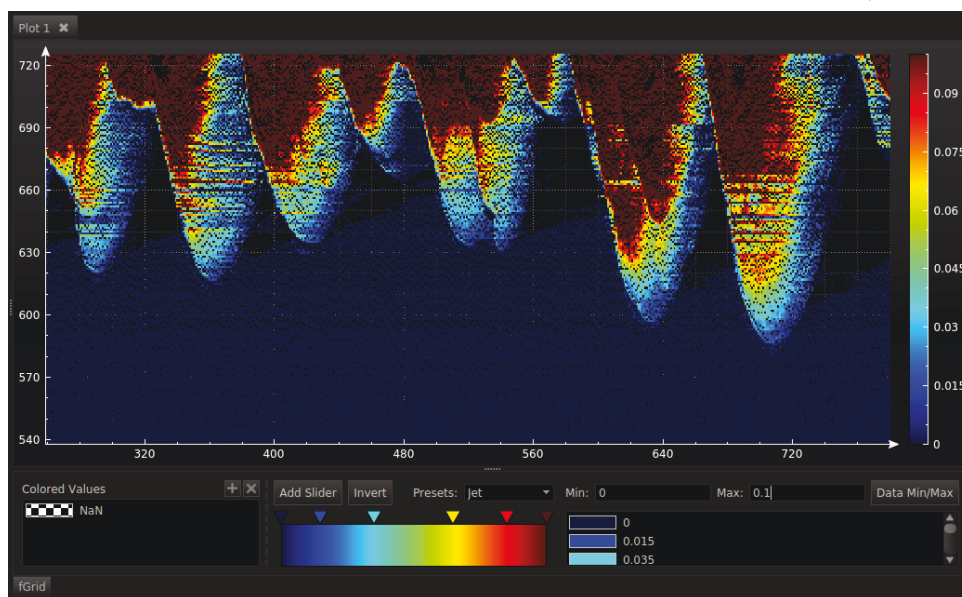
- **Add to** – the variable can be plotted on a new plot or added to any existing plot.
- **Plot Type** – choose the type of plot. Currently only PColor is available.
- **Section Size** – the number of columns per plotted section.
- **Section Overlap** – the number of columns to overlap when toggling between sections of the plot.
- **Section to Display** – the section to show when this dialog is closed.
- **Invert Axes** – choose this option to invert the rows and columns of this data before plotting.

The plot section settings are a key component in visualizing runup data. Because the data are typically much longer in the x-axis (time) than in the

y -axis (cross-shore distance), a single plot containing the entire linescan displayed on a screen will be very compressed in the x -direction, making it difficult to distinguish the runup line. Therefore, it is beneficial to view smaller sections of the x -axis and toggle through sections as needed. These settings determine how much of the x -axis data will be displayed in each section. When viewing the data, the user can move through the sections by clicking the left and right arrows of the plot section chooser.

Once a variable has been plotted, a number of options are available to the user to better visualize the data. As an example, the plot of the *fGrid* variable in Figure 5-6 will be used in describing these options.

Figure 5-6. A *RunupTool* plot of the *fGrid* variable, which is the product of the interpolated foreshore grid being subtracted from the elevation grid (*zGrid*). Warmer colors represent greater differences between the variables (deeper water). This product allows for clearer visualization of the swash elevations above the dry beach.



- **Pan and zoom** – the left mouse button can be used to pan around the plot by clicking and dragging, and the scroll wheel can be used to zoom in and out of the plot. Additionally, the individual axes can be scaled by first clicking on the axis to select it (turning it blue) and using the scroll wheel to scale. Clicking anywhere on the plot will deselect the axis.
- **Colored values** – by default, any value in the variable that is a NaN will be completely transparent on the plot. Additional specifically colored values can be added using the “plus” button in the *Colored Values* pane at the bottom of the plot. Colored values can be removed by selecting them and clicking the “X” button.

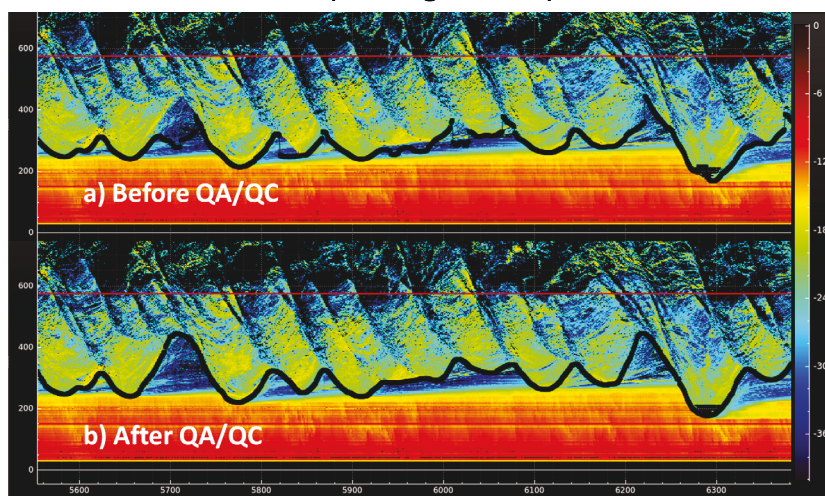
- **Color gradients** – the color of each plotted value is determined using the color gradient displayed below the plot, and there are numerous ways to manipulate the gradient. Use the triangular sliders to adjust the colorbar. As the sliders are moved, the value associated with the slider is reflected in the list to the right of the gradient. Values can be manually entered into the list by double-clicking the list item. The color of a slider can be changed by single-clicking the slider. Sliders can be added using the “Add Slider” button, and sliders can be removed by right-clicking the slider and choosing “Remove Slider.” A number of preset gradients are provided in the “Presets” dropdown menu. Another useful tool is the “Data Min/Max” button, which will map the minimum and maximum values of the data to the leftmost and rightmost sliders, respectively, and interpolate the values for the remainder of the sliders based on their location in the gradient. The gradient scale located to the right of the plot can be dragged and zoomed, which will manipulate the “Min” and “Max” values of the gradient in a similar fashion.

As variables are added to the same plot, the options associated with each variable are added to the tabs at the absolute bottom of the plot view.

5.2.6 Specifying existing runup line

To visualize and edit the runup line, the user must select which variable in the current dataset contains the runup line data. In the FRF’s current workflow, this variable is `lineRunupCalc/downLineIndex`. The variable is set as the runup line by locating the variable in the variable display, right-clicking, and choosing “Set as Runup Line” from the context menu. When the runup line is set, all plots will be updated to include the runup line, as shown in Figure 5-7, and all editing tools will become available. At this point, the user is ready to begin editing the runup line.

Figure 5-7. The runup lines (black line) displayed on *RunUpTool*/agrid (lidar return amplitude data) plots (a) before and (b) after QA/QC. Warmer colors represent greater amplitude values.



5.2.7 Simple editing tools

RunupTool provides two simple editing tools which can be used to directly manipulate the runup line. These tools, snap-to-click and snap-to-drag, can be used by clicking the “S” or “D” tool buttons, respectively, from the editing toolbar. Edits to the runup line are done using the right mouse button, leaving the left mouse button available for navigating around plots.

- **Snap-to-click** – right click anywhere on a plot while this tool is active and the runup line at the x -location of the mouse will be moved to the mouse location.
- **Snap-to-drag** – right click and drag on a plot while this tool is active and the runup line will snap to the mouse location as it moves along the x -axis.

Any edits made to the runup line can be undone by clicking the undo button, located on the program toolbar.

5.2.8 Algorithmic editing tools

In addition to the simple direct editing tools, *RunupTool* provides three algorithmic editing tools that can be used to manipulate the runup line. As opposed to direct user manipulation of the runup line, these tools make use of existing data, along with a few user specified values, to recalculate the entire runup line at once. Each of the three tools performs calculations using the entire dataset and thus makes changes to the runup line as a

whole. All three tools can be controlled from the editing toolbar. Each of the following descriptions has the available user specified values in *italics*.

- **Runup Threshold Tool** – for each scan line, places the runup line at the cross-shore location of the first instance of a value above the set *runup threshold*. Searching for this location will be contained to the area between *min swash* and *max swash*, both of which are indicated on all plots by a horizontal red line. The variable containing searchable values is set using the *apply-to* option. As an example, to place the runup line at the first location where the value of the *zDiff* variable is greater than 10 cm, and the user knows the runup line falls in the cross-shore indices of 600 and 700, the values would be set as follows:
 - *Min Swash* – 600
 - *Max Swash* – 700
 - *Runup Threshold* – 0.1
 - *Apply to* – *zGrid*

Note that for a variable to appear in the *apply to* dropdown menu, it must first be loaded into a plot.

- **Recalculate *fGrid* Tool** – the recalculate *fGrid* tool recalculates the *fGrid* variable based on edits that have been made to the runup line. The *fGrid* variable, which is initially calculated in the MATLAB processing workflow, is the elevation grid (*zGrid*) minus the foreshore grid. This foreshore grid is calculated by removing the “water” data from the *zGrid* variable, based on the location of the runup line, and filling gaps in the data by interpolating through time. Therefore, when edits are made to the runup line using *RunupTool*, the *fGrid* variable needs to be recalculated. Set the *fGrid* option to the *fGrid* data and the *zGrid* option to the *zGrid* data and press *Recalculate* to recalculate the *fGrid* variable for the entire dataset. Note that for the *fGrid* and *zGrid* variables to appear in the option menus, they must both be loaded onto a plot.
- **Median Filter Tool** – applies a median filter to the entire runup line. For each scan line, the filter looks at the location of the runup line in adjacent scan lines and sets the location of the current scan line to the median value. For example, with a *window size* of 5, the location of the runup line will be set to the median of the runup location of the current scan line, the two previous scan lines, and the two subsequent scan lines.

5.2.9 Saving and committing edits

When a user has either finished using *RunupTool* and wishes to save changes made to the current runup line or has finished editing a runup and wishes to send the data to be reprocessed, they must either save or commit their changes. The differences between saving and committing are as follows:

- **Saving** changes will save the edited runup line so that when the dataset is reopened in *RunupTool*, the edits made to the runup line will be present. Saving is done by clicking the save icon in the toolbar. When the data has been saved, a loading icon will appear next to the file in the file viewer.
- **Committing** changes will, in addition to performing the tasks performed when saving the data, will write a new variable to the data file containing the new runup line, as well as QA/QC data, which includes the user's name, email, and organization. The dataset is then placed into the reprocessing queue so that the automated workflow will reprocess the dataset using the new runup line. When a dataset has been successfully committed, a green checkmark will appear next to the file in the file viewer. Committing is done by clicking the "Commit Changes" button at the bottom of the variable viewer.

After a dataset has been saved or committed, *RunupTool* can be closed or a new dataset can be opened, and all edits that have been made to the current dataset will persist.

6 Data Products

The following sections describe in detail the data products that are generated from the real-time data processing workflow described in Section 4.

6.1 Linescan data products

Linescan data products are saved in the **.line.science.mat* file within each collect's hourly folder and are computed using a series of scripts:

DoMorphologyScience.m, *DoRunupScience.m*,
DoHydrodynamicScience.m, and *DoWaveGaugeScience.m*.

These scripts are called within *ProcessLinescan.m* which can be found here:

DuneLidar/Linescan/ProcessLinescan.m

Select linescan data products are also converted to NetCDF files and available on the CHL THREDDS server: <http://chlthredds.erdcdren.mil/> and associated GUI: <https://frfdataportal.erdcdren.mil/>. A brief description of all data products will be provided below. The NetCDF file generation process is summarized in Section NetCDF Generation for linescan and framescan data products. Please refer to Appendix I for specific variable names and details on the variable structs.

6.1.1 Morphology

Hourly linescan morphology products describe the topography of the beach on the narrow cross-shore transect scanned by the lidar and are stored in the `lineMorphologyData` struct. These data products are calculated in the script *DoMorphologyScience.m*, which can be found here:

DuneLidar/Linescan/Morphology/DoMorphologyScience.m

Each product is briefly described below:

- **Mean Foreshore** – the mean elevation of the beach at each location in the cross-shore profile at 10cm increments from the lidar (defined by the `downLinePosition` variable) during the 30-minute linescan.
- **Min Foreshore** – the minimum elevation of the beach at each location in the cross-shore profile at 10 cm increments from the lidar (defined by the `downLinePosition` variable) during the 30-minute linescan.
- **Max Foreshore** – the maximum elevation of the beach at each location in the cross-shore profile at 10 cm increments from the lidar (defined by the `downLinePosition` variable) during the 30-minute linescan.
- **Foreshore Standard Deviation** – the standard deviation in elevation of the beach at each location in the cross-shore profile at 10 cm increments from the lidar (defined by the `downLinePosition` variable) during the 30-minute linescan. The standard deviation is included to provide the user with an assessment of how much the profile evolved during the 30-minute linescan. During large waves, elevation changes on the order of tens of centimeters are not uncommon within the active swash zone.
- **Curvature** – a 5 m running average of the instantaneous curvature of the beach profile at 10 cm increments from the lidar (defined by the `downLinePosition` variable). Calculated using *curv1DSmooth.m*.
- **Curvature 2** – the curvature of the beach calculated over a 5 m region at 10 cm increments from the lidar (defined by the `downLinePosition` variable). Calculated using *curv1DSkipPoints.m*.
- **Instantaneous Slope** – the instantaneous cross-shore slope at 10 cm increments from the lidar (defined by the `downLinePosition` variable).
- **Foreshore Slope** – the slope of the beach in the swash zone, calculated as the slope of the best-fit linear trend to the beach elevation profile in the region defined by the mean horizontal swash position ± 2 standard deviations σ in the swash.
- **Foreshore Slope R^2** – the r-squared coefficient of the linear fit to the beach foreshore.

A selection of the morphology data products are then converted to variables in a NetCDF file and stored on the CHL THREDDS server. The Morphology NetCDF file contains the mean foreshore dry beach elevation, the standard deviation in foreshore elevations, and the foreshore slope as

described above. The grid for mean foreshore and foreshore standard deviation is included in two different formats: FRF cross-shore, alongshore and downline distance from the lidar scanner (lidar location supplied in latitude, longitude and FRF cross-shore, alongshore). As a quality indicator, the Beach Profile QC Flag indicates the longest temporal range in dry beach elevation data at each cross-shore bin where there are continuous data at a rate of at least 2Hz.

Data gaps in the foreshore elevation occur when the waves run up and down the beach. The total percentage of dry beach data at each cross-shore location used in the calculation of mean, minimum, maximum, and standard deviation is also reported.

6.1.2 Wave runup

Hourly wave runup data products describe the time-series of the position and elevation of the instantaneous shoreline (wave runup) during each linescan and are stored in the `lineRunupData` structure. These products are calculated by the *DoRunupScience.m* script located here:

DuneLidar/Linescan/Runup/DoRunupScience.m

Note that the wave runup data products and associated statistics provided here reflect the total water elevation in meters, NAVD88 and contributions from tide and surge have not been removed. The runup time-series and associated statistics are provided in both the horizontal and vertical. Horizontal coordinates are provided as the cross-shore distance from the lidar (*downLineDistance*) and can be related to FRF coordinates using the *downLineX* and *downLineY* variables provided in *lineCoredat*. Vertical statistics are reported in meters, NAVD88.

- **Cross-Shore**

- **Minimum** – cross-shore position of the minimum rundown over the 30-minute time-series. This can be used to denote the offshore edge of the swash zone or the effective lower limit or *base* of the foreshore.
- **Maximum** – cross shore position of the maximum runup over the 30-minute time-series. This location can be used to denote the onshore edge of the swash zone or the effective upper limit of wave inundation,

- **Mean** – the average cross-shore position of wave runup over the 30-minute time-series. This location is used to denote the mean position of the swash zone.
- **Standard Deviation** – the standard deviation in the cross-shore swash excursion over the 30-minute time-series.
- **R2%** – the cross-shore location that is exceeded only 2% of the time by wave runup (e.g., Stockdon et al. 2006). This statistic is a more stable representation of the maximum onshore extent of wave runup during the time-series.
- **Time-Series** – the cross-shore position of wave runup time-series during the 30-minute linescan.
- **Time-Series Index** – the time-series of indexed cross-shore position of wave runup during the 30-minute linescan. This index can be used to extract the time-series in FRF coordinates if desired.
- **Spectral Analysis**
 - Frequency variable
 - Swash Excursion Amplitude Spectrum – Power Spectral Density estimate of the swash excursion distance using an 8-minute window with 50% overlap.
 - Significant Infragravity Swash Excursion Distance – significant swash excursion distance for $f < 0.05$.
 - Significant Sea/Swell Swash Excursion Distance - Significant swash excursion distance for $0.05 < f < 0.2$.
- **Vertical**
 - **Minimum** – elevation of the minimum rundown over the 30-minute time-series. This can be used to identify the elevation at the offshore edge of the swash zone or the effective elevation of the base of the foreshore.
 - **Maximum** – elevation of the maximum runup over the 30-minute time-series. This location can be used to denote the highest elevation reach by wave runup.
 - **Mean** – the average elevation of wave runup over the 30-minute time-series. Once tide and surge are subtracted this elevation is equivalent to the mean swash, or wave setup elevation in Stockdon et al. (2006).
 - **Standard Deviation** – the standard deviation in the elevation of swash over the 30-minute time-series.

Multiplying this value by 4 would provide a significant swash height.

- **R2%** – the elevation that is exceeded only 2% of the time by wave runup events. This statistic is a more stable representation of the maximum onshore extent of wave runup during the time-series. Once tide and surge are subtracted, this elevation is equivalent to the R2% elevation predicted by Stockdon et al. (2006).
- **Time-Series** – the time-series of the elevation of wave runup during the 30-minute linescan.
- **Spectral Analysis**
 - Frequency variable
 - Swash Elevation Amplitude Spectrum - Power Spectral Density estimate of the swash elevation using an 8-minute window with 50% overlap.
 - Significant Infragravity Swash Height – significant swash height for $f < 0.05$.
 - Significant Sea/Swell Swash Height – significant swash height for $0.05 < f < 0.2$.

A selection of the runup statistics and time-series data products are then converted to NetCDF files and stored on the CHL THREDDS server. The Wave Runup NetCDF file, *FRF-ocean_waves_lidarWaveRunup_[year][month].nc*, contains the vertical R2% as described above along with the full runup line time-series in three different formats: in FRF cross-shore, alongshore, in downline distance from the lidar scanner (lidar location supplied in latitude, longitude and FRF cross-shore, alongshore), and in vertical elevation. For quality assurance, the percentage of the time-series that is missing is reported along with the Total Water Level Flag to indicate that real-time runup data have passed essential standards but have not formally been QA/QC'd.

6.1.3 Hydrodynamics

Hourly hydrodynamic data products are derived from the time-series of water surface elevation data collected by the lidar and are stored in the `lineHydrodynamicsData` structure. These products are calculated by the *DoHydrodynamicScience.m* script located here:

DuneLidar/Linescan/Hydrodynamics/DoHydrodynamicScience.m

Hydrodynamic data products are based on only the portion of the points that reflected off water during the 30-minute scan (i.e., the portion of the each linescan offshore of the runup line). All hydrodynamic statistics are calculated on the interpolated time-series that are gridded at 10 cm increments from the lidar, effectively generating a *wave gauge* every 10 cm in the cross-shore. Therefore, cross-shore profiles of water level and wave statistics are provided as the final products. The offshore extent of these products is dependent on the wave conditions at the time of linescan. The larger the waves and the wider the inner surf-zone, the farther offshore that these products will extend. During calm, low-wave conditions, these products may have very short ranges in the cross-shore.

- **Mean Water Level** – the mean elevation of the water surface at each location in the cross-shore profile at 10cm increments from the lidar (defined by the `downLinePosition` variable) during the 30-minute linescan. Can be used to calculate wave setup at any location along the cross-shore profile.
- **Min Water Level** – the minimum elevation of the water surface at each location in the cross-shore profile at 10cm increments from the lidar (defined by the `downLinePosition` variable) during the 30-minute linescan.
- **Max Water Level** – the maximum elevation of the water surface at each location in the cross-shore profile at 10 cm increments from the lidar (defined by the `downLinePosition` variable) during the 30-minute linescan.
- **Water Level Standard Deviation** – the standard deviation in the elevation of the water surface at each location in the cross-shore profile at 10 cm increments from the lidar (defined by the `downLinePosition` variable) during the 30-minute linescan.
- **H_{sig}** – the significant wave height, calculated as 4* (the standard deviation in the elevation of the water surface), at each location in the cross-shore profile at 10 cm increments from the lidar (defined by the `downLinePosition` variable) during the 30-minute linescan.
- **FFT** – spectral analysis at each location in the cross-shore profile at 10 cm increments from the lidar (defined by the `downLinePosition` variable) during the 30-minute linescan.
 - Frequency Variable
 - Amplitude – Power Spectral Density estimate of the wave energy using an 8-minute window with 50% overlap.

- Significant Infragravity Wave Height – the significant wave height in the infragravity band ($f < 0.05$ Hz)
- Significant Sea/Swell Wave Height – the significant wave height in the sea/swell band ($0.05 \text{ Hz} < f < 0.2 \text{ Hz}$)
- Peak Infragravity Period – the period with the highest energy at each cross-shore location in the infragravity band ($f < 0.05 \text{ Hz}$)
- Peak Sea/Swell Period – the period with the highest energy at each cross-shore location in the sea/swell band ($0.05 \text{ Hz} < f < 0.2 \text{ Hz}$)
- **Wave Skewness** – the skewness of the wave time-series at each location in the cross-shore profile at 10 cm increments from the lidar (defined by the `downLinePosition` variable) during the 30-minute linescan. Positive values mean waves have narrow, peaked crests and wide, broad troughs.
- **Wave Asymmetry** – the asymmetry of the wave time-series at each location in the cross-shore profile at 10 cm increments from the lidar (defined by the `downLinePosition` variable) during the 30-minute linescan. Wave asymmetry is calculated as the skewness of the imaginary component of the hilbert transform of the surface elevation time-series (Elgar and Guza 1985). Positive values describe waves with steep, pitched forward faces and lower sloped backs.

6.1.4 Wave gauge

Hourly wave gauge data products provide wave time-series and associated statistics and spectral analysis at a sub-set of cross-shore locations in the field of view of the lidar.

A selection of the wave gauge statistics and time-series data products are then converted to NetCDF files and stored on the CHL THREDDS server. A description of the wave gauge netcdf files follows below:

The Wave Gauge NetCDF files, *FRF-ocean_waves_lidarWaveGauge* *[cross-shore coord]_[year][month].nc*, are virtual wave gauges at 80 m, 90 m, 100 m, 110 m, and 140 m in FRF cross-shore, alongshore, in latitude, longitude, and in downline, offline distance from the lidar scanner. Each Wave Gauge NetCDF file reports the virtual wave gauge's entire time-series from which traditional wave gauge, and spectral statistics are derived along with the percentage of the total time-series with no measurements as well as the median duration of the time gap

where there are no measurements (in seconds) . The wave gauge statistics include mean water level, maximum water level, minimum water level, and significant wave height. Flags indicate the confidence in the associated wave gauge statistics. For quality-assurance purposes, a good flag signifies that the 30-minute scan was sampled at an average of 2 Hz or higher throughout the scan, a bad flag indicates 1 to 80 seconds of sampling at 2 Hz or higher, with a questionable flag labeling a scan as falling in between (2Hz or greater for 80 seconds to 30 minutes). The spectral statistics include wave frequency, amplitude, significant infragravity wave height, significant sea/swell wave height, peak infragravity period, and peak sea/swell period as described above. Spectral wave gauge statistics are also flagged to signify data quality, with good, questionable, and bad flags corresponding to the longest consecutive scan period of over 29 minutes, 17–29 minutes, and fewer than 17 minutes of sampling at or over 2 Hz.

6.2 Framescan data products

Once the initial framescan has been generated from the SOCS, a number of associated data products are generated. These data products are saved in the **.frame.data.mat* file within each collect's hourly folder. In its current form this includes the following structures:

`frameGriddedData`

- `data` – the final bare earth DEM, generated after the filtered point cloud is interpolated onto a regular 0.1 m grid
- `xs` – the cross-shore X vector defining the DEM grid, which spans from $x = 39.975$ m to $x = 179.975$ m with 0.1 m spacing
- `as` – the alongshore Y vector defining the DEM grid, which spans from $y = 399.975$ m to $y = 1199.975$ m with 0.1 m spacing
- `waterLine` – the x-coordinate of the alongshore location of the water line for each lidar phi angle, determined using the *waterfilter.m* function
- `waterLineStd` – the standard deviation of the location of the water line
- `reflectanceData` – the reflectance data from the hourly scan, interpolated onto the same regular grid.
- `info` – additional information about the grid, including the north, south, east, and west extent of the grid, the number of columns and rows in the grid, and the grid resolution.

`frameRawRectifiedData`

- *filenames*
 - `name` – the names of the *.rxp* files used to generate the bare earth DEM
 - `date` – the time and date these files were imported, in Day-Month-Year Hr:Min:Sec format
 - `bytes` – the size of the *.rxp* files
 - `datenum` – the date number of time the files were imported

`frameFinishTime`: the time the frame scan processing finished.

6.3 Beach digital elevation model (DEM)

The DEM is an hourly bare earth elevation model of the dry beach north of the FRF pier. The surface is created during `ProcessFramescan` (found at the location listed below) and data are stored in the `frameGriddedData` structure:

DuneLidar/Framescan /ProcessFramescan.m .

The variable is briefly described as follows:

- **Bare-earth Sub-aerial Beach Elevation Grid** – Gridded array of bare-earth, sub-aerial beach elevations. Data have been filtered to remove water, vegetation, and other non-ground points. Filtering was performed with an automated filter but has not been manually reviewed. Each point's location in latitude, longitude, and FRF coordinates is found at the same indices in the `frameGriddedData` structure.

The Beach DEM is then written into a NetCDF file and stored on the CHL THREDDS server. The Beach DEM NetCDF file contains the bare-earth, sub-aerial beach elevations as described above. The grid for the beach elevations is included in two different formats: FRF cross-shore, alongshore, and latitude, longitude. The grid point locations in FRF coordinates are included as 1D vectors to reduce file size. The grid rows share an alongshore coordinate and the columns share a cross-shore coordinate. The grid point locations in latitude and longitude are stored as 2D arrays where each value in the matrix is the coordinate for the elevation at the same indices. The elevation model is not manually reviewed, and the NetCDF indicates such.

6.4 NetCDF Generation for linescan and framescan data products

The selected Data Product NetCDF files associated with each data product are each created by building a structure aligned to a human-readable data serialization language (YAML) template file, then that structure is written into a NetCDF file, along with the metadata, using a suite of MATLAB to NetCDF utility functions. The metadata and data to be included in the NetCDF file are defined in the YAML template files. A guide for required and recommended metadata is found in the metadata directory and the data template file can be tailored to suit desired data products. The NetCDF file is written using *matlab2netcdf.m* and other utility functions located in *DuneLidar/FDIF/utilities*. All actions and messages from the utility functions are reported to MATLAB's Command Window and in a log file created in the present working directory. The *matlab2netcdf.m* function requires a structure of data, a matching template file, a metadata description file, and an output file handle.

Table 6-1 – Table 6-5 list the NetCDF files generated for morphology, wave runup, hydrodynamics, and wave gauge data products based on both a single hourly scan and an entire month where each scan has passed QA/QC. During real-time processing, the MATLAB structure of data as described in the YAML template is assembled, followed by calling utility functions at the end of saving functions, which also build the final NetCDF file handles and convert any nested MATLAB datenums within each structure to seconds since epoch (non-nested datenums are converted within *matlab2netcdf.m*). Table 6-1 – Table 6-5 also list each structure assembly function and saving function.

Monthly NetCDF files are controlled by *MonthlyNetCDFGeneration.m*, located here:

DuneLidar/FDIF/MonthlyNetCDFGeneration.m

MonthlyNetCDFGeneration.m assures that all QA/QC'd, reprocessed NetCDF files within a given month are present before passing the scan directories to respective structure assembly functions that will aggregate data to be written into the monthly NetCDF file (Table 6-1 – Table 6-5). Each script loads the needed structures from the ordered *line.science.mat* files and appends all data to the end of a single monthly structure to be written into the monthly NetCDF file in proper temporal order. As with the other data product generation files, utility functions

are called at the end of each monthly saving function following structure assembly (Table 6-1 – Table 6-5).

Table 6-1. Linescan NetCDF generation files – morphology.

Linescan NetCDF Generation Files – Morphology	
Final NetCDF File Handle (real-time and monthly)	<i>FRF-geomorphology_elevationTransects_duneLidarTransect_[year][month][day]_[hour][minute][seconds].nc</i> <i>FRF-geomorphology_elevationTransects_duneLidarTransect_[year][month].nc</i>
YAML Template Files (metadata and data)	<i>DuneLidar/FDIF/metadata/linescan_beach_profile_metadata.yml</i> and <i>DuneLidar/FDIF/templates/linescan_beach_profile.yml</i> .
MATLAB structure assemblies (real-time and monthly)	<i>MakeBeachProfileNetCDFStruct.m</i> at the end of <i>ProcessLinescan.m</i> using the <i>lineCoredat</i> and <i>lineMorphologyData</i> structures; <i>MakeBeachProfileNetCDFStructMonthly.m</i>
Save Functions (real-time and monthly)	<i>SaveBeachProfileMetadata.m</i> <i>SaveBeachProfileMetadataMonthly.m</i>
Location – Structure Assemblies/Saving Functions (real-time)	<i>DuneLidar/Linescan/Morphology/SaveData/</i>

Table 6-2. Wave runup NetCDF generation files.

Wave Runup NetCDF Generation Files	
Final NetCDF File Handle (real-time and monthly)	<i>FRF-ocean_waves_lidarWaveRunup_[year][month][day]_[hour][minute][seconds].nc</i> <i>FRF-ocean_waves_lidarWaveRunup_[year][month].nc</i>
YAML Template Files (metadata and data)	<i>DuneLidar/FDIF/metadata/linescan_wave_runup_metadata.yml</i> and <i>DuneLidar/FDIF/templates/linescan_wave_runup.yml</i> , respectively.
MATLAB structure assemblies (real-time and monthly)	<i>MakeWaveRunupNetCDFStruct.m</i> at the end of <i>ProcessLinescan.m</i> using the <i>lineCoredat</i> and <i>lineRunupData</i> structures; <i>MakeWaveRunupNetCDFStructMonthly.m</i>
Save Functions (real-time and monthly)	<i>SaveWaveRunupMetadata.m</i> <i>SaveWaveRunupMetadataMonthly.m</i>
Location – Structure Assemblies/Saving Functions (real-time)	<i>DuneLidar/Linescan/Runup/SaveData/</i>

Table 6-3. Hydrodynamics NetCDF generation files.

Hydrodynamics NetCDF Generation Files	
Final NetCDF File Handle (real-time and monthly)	<i>FRF-ocean_waves_lidarHydrodynamics_[year][month][day]_[hour][minute][seconds].nc</i> <i>FRF-ocean_waves_lidarHydrodynamics_[year][month].nc</i>
YAML Template Files (metadata and data)	<i>DuneLidar/FDIF/metadata/linescan_hydrodynamics_metadata.yml</i> and <i>DuneLidar/FDIF/templates/linescan_hydrodynamics.yml</i> , respectively.
MATLAB structure assemblies (real-time and monthly)	<i>MakeHydrodynamicNetCDFStruct.m</i> at the end of <i>ProcessLinescan.m</i> using the <i>lineCoredat</i> and <i>lineHydrodynamicsData</i> structures; <i>MakeHydrodynamicNetCDFStructMonthly.m</i>
Save Functions (real-time and monthly)	<i>SaveHydrodynamicsMetadata.m</i> <i>SaveHydrodynamicsMetadataMonthly.m</i>
Location – Structure Assemblies/Saving Functions (real-time)	<i>DuneLidar/Linescan/Hydrodynamics/SaveData/</i>

Table 6-4. Wave gauge linescan NetCDF generation files.

Wave Gauge Linescan NetCDF Generation Files	
Final NetCDF File Handle (real-time and monthly)	<i>FRF-ocean_waves_lidarWaveGauge[cross-shore coord]_[year][month][day]_[hour][minute][seconds].nc</i> <i>FRF-ocean_waves_lidar WaveGauge[cross-shore coord]_[year][month].nc</i>
YAML Template Files (metadata and data)	<i>DuneLidar/FDIF/metadata/linescan_wave_gauge_metadata_[cross-shore coord].yml</i> and <i>DuneLidar/FDIF/templates/linescan_wave_gauge.yml</i>
MATLAB structure assemblies (real-time and monthly)	<i>MakeWaveGaugeNetCDFStruct.m</i> at the end of <i>ProcessLinescan.m</i> using the <i>lineWaveGaugeData</i> structure; <i>MakeWaveGaugeNetCDFStructMonthly.m</i>
Save Functions (real-time and monthly)	<i>SaveWaveGaugeMetadata.m</i> <i>SaveWaveGaugeMetadataMonthly.m</i>
Location – Structure Assemblies/Saving Functions (real-time and monthly)	<i>DuneLidar/Linescan/WaveGauges/SaveData/</i>

Table 6-5. Beach DEM framescan NetCDF generation files.

Framescan NetCDF Generation – Beach Digital Elevation Model	
Final NetCDF File Handle (real-time and monthly)	<i>FRF-geomorphology_DEMs_duneLidarDEM_ [year][month][day]_[hour][minute][seconds].nc</i> <i>FRF-geomorphology_DEMs_duneLidarDEM_[year][month].nc</i>
YAML Template Files (metadata and data)	<i>DuneLidar/FDIF/metadata/framescan_beach_DEM_metadata.yml</i> and <i>DuneLidar/FDIF/templates/framescan_beach_DEM.yml</i>
MATLAB structure assemblies (real-time and monthly)	<i>MakeDEMNetCDFStruct.m</i> at the end of <i>ProcessFramescan.m</i> using the <i>frameGriddedData</i> , <i>allFramescans</i> , and <i>dataRectification</i> structures; <i>MakeDEMNetCDFStructMonthly.m</i>
Save Functions (real-time and monthly)	<i>SaveDEMeMetadata.m</i> <i>SaveDEMMetadataMonthly.m</i>
Location – Structure Assemblies/ Saving Functions (real-time and monthly)	<i>DuneLidar/Framescan/DEM/</i>

7 Conclusion

The installation and continued operation of the dune-mounted terrestrial lidar scanner adds a new dimension to the rich suite of datasets capturing the evolution of the dynamic coastal environment at ERDC-CHL's FRF. The continuous dune-mounted lidar tower provides high spatial (centimeter-scale) and temporal resolution (seconds to minutes) data on beach morphology and shoreline and inner-surf-zone hydrodynamics. The methodology described in this document provides robust tools to automatically manage and process these data in near real-time through a range of different conditions. This ERDC-CHL SR has been designed as an internal reference manual documenting the equipment, procedures, and code used to collect and process these data. This report should also be utilized by users of the data who are looking for a more in-depth description of the data products and processing methodology.

The methodologies described in this report document the efficient storage, processing, evaluation, and management of the large amount of high density point clouds generated by the lidar scans. Care has been taken to develop processing approaches that are automated and robust to a variety of environmental conditions; however, the user should be aware that they are using a Research and Development (R&D) level data product. As with any R&D system, processing methodologies are continually being improved and updated. Therefore, this report should be considered up to date only through April 2017. Updates to the code and changes will be published as necessary in follow-on special reports. Future updates to the physical system will focus on improving the protective housing to reduce maintenance and cleaning intervals. Updates to data management and storage problems will explore new data compression techniques and the ability to utilize cloud based services for data storage. Updates to data pre-processing may include changes to the coregistration algorithm or wave runup extraction. In particular, efforts will be made to reduce the need for user intensive QA/QC via *RunupTool*, by improving the automated function for finding the instantaneous shoreline position in each linescan.

The ability to capture near-real-time coastal morphological evolution and quickly disseminate the data through a suite of datasets and useable products opens up the potential for developing critical analysis and decision-making tools for USACE, the Army, and other coastal stakeholders operating in environments with dynamic topographic

changes. The use of lidar provides these observations through conditions where other data collection methods fail to allow safe operation or adequately robust data/data interpretation. The continued lidar tower operation therefore greatly increases the ability to quantify forcing and response during coastal storms and other extreme events, capturing data in the nearshore region with a temporal and spatial frequency which has previously been unattainable. These data will be used to validate and develop coastal numerical models as part of the Coastal Model Testbed at the FRF and integrated into an International Coastline Observatory Network website. The support of model evaluation and development will lead to improved forecasting of future coastal inundation and shoreline change scenarios. As a result, USACE will be more prepared to manage flood risks during extreme events and better engineer coastal infrastructure for future coastal flood risks.

References

- Aagaard, T., and B. Greenwood. 1994. "Suspended Sediment Transport and the Role of Infragravity Waves in a Barred Surf Zone." *Marine Geology* 118(1-2): 23–48.
- Almeida, L. P., G. Masselink, P. Russell, M. Davidson, T. Poate, R. McCall, C. Blenkinsopp, and I. Turner. 2013. "Observations of the Swash Zone on a Gravel Beach during a Storm Using a Laser-Scanner (Lidar)." *Journal of Coastal Research* 65(sp1): 636–641.
- Blenkinsopp, C. E., M. A. Mole, I. L. Turner, and W. L. Peirson. 2010. "Measurements of the Time-Varying Free-Surface Profile across the Swash Zone Obtained Using an Industrial LIDAR." *Coastal Engineering* 57(11): 1,059–1,065.
- Blenkinsopp, C. E., I. L. Turner, M. J. Allis, W. L. Peirson, and L. E. Garden. 2012. "Application of LiDAR Technology for Measurement of Time-Varying Free-Surface Profiles in a Laboratory Wave Flume." *Coastal Engineering* 68: 1–5.
- Brodie, K. L., B. Raubenheimer, S. Elgar, R. K. Slocum, and J. E. McNinch. 2015. "Lidar and Pressure Measurements of Inner-Surfzone Waves and Setup." *Journal of Atmospheric and Oceanic Technology* 32(10): 1,945–1,959.
- Brodie, K. L., R. K. Slocum, and J. E. McNinch. 2012. "New Insights into the Physical Drivers of Wave Runup from a Continuously Operating Terrestrial Laser Scanner." In *Oceans 2012*: 1–8. IEEE.
- Eamer, J. B., and I. J. Walker. 2013. "Quantifying Spatial and Temporal Trends in Beach–Dune Volumetric Changes Using Spatial Statistics." *Geomorphology* 191: 94–108.
- Forte, Michael F., William A. Birkemeier, and J. Robert Mitchell. 2017. *Nearshore Survey System Evaluation*. ERDC/CHL SR-17-19. Vicksburg, MS: U.S. Army Engineer Research and Development Center.
- Hanson, J. L., H. C. Friebel, and K. K. Hathaway. 2009. "Coastal Wave Energy Dissipation: Observations and STWAVE-FP Performance." In *11th International Workshop on Wave Hindcasting and Forecasting and 2nd Coastal Hazards Symposium*, 18–23.
- Holman, R. A. 1986. "Extreme Value Statistics for Wave Run-Up on a Natural Beach." *Coastal Engineering* 9(6): 527–544.
- Holman, R., N. Plant, and T. Holland. 2013. "cBathy: A Robust Algorithm for Estimating Nearshore Bathymetry." *Journal of Geophysical Research: Oceans* 118(5): 2,595–2,609.
- Houser, C., and B. Greenwood. 2007. "Onshore Migration of a Swash Bar during a Storm." *Journal of Coastal Research* 23(1): 1–14.
- Houser, C., B. Greenwood, and T. Aagaard. 2006. "Divergent Response of an Intertidal Swash Bar." *Earth Surface Processes and Landforms* 31(14): 1,775–1,791.

- Howd, P. A., and R. A. Holman. 1987. "A Simple Model of Beach Foreshore Response to Long-Period Waves." *Marine Geology* 78(1-2): 11–22.
- Kobayashi, N., and H. Jung. 2012. "Beach Erosion and Recovery." *J. Waterw. Port Coastal Ocean Eng.* 138(6): 473–483. doi:10.1061/(ASCE)WW.1943-5460.0000147.
- Lippmann, T. C., and R. A. Holman. 1990. "The Spatial and Temporal Variability of Sand Bar Morphology." *Journal of Geophysical Research* 95(C7): 11,575.
- Lippmann, T. C., and R. A. Holman. 1989. "Quantification of Sand Bar Morphology: A Video Technique Based on Wave Dissipation." *Journal of Geophysical Research* 94(C1): 995–1,011.
- Long, Charles E., and Joan M. Oltman-Shay. 1991. *Directional Characteristics of Waves in Shallow Water*. CERC-TR-91-1. Vicksburg, MS: Waterways Experiment Station, Coastal Engineering Research Center.
- Masselink, G., and S. van Heteren. 2014. "Response of Wave-Dominated and Mixed-Energy Barriers to Storms." *Marine Geology* 352: 321–347.
- Pianca, C., R. Holman, and E. Siegle. 2015. "Shoreline Variability from Days to Decades: Results of Long-Term Video Imaging." *Journal of Geophysical Research: Oceans* 120(3): 2,159–2,178.
- Pietro, L. S., M. A. O'Neal, and J. A. Puleo. 2008. "Developing Terrestrial-LIDAR-Based Digital Elevation Models for Monitoring Beach Nourishment Performance." *Journal of Coastal Research* 24(6): 1555–1564.
- Puleo, J. A., T. Lanckriet, and C. Blenkinsopp. 2014. "Bed Level Fluctuations in the Inner Surf and Swash Zone of a Dissipative Beach." *Marine Geology* 349: 99–112.
- Roelvink, D., A. Reniers, A. P. Van Dongeren, J. V. T. de Vries, R. McCall, and J. Lescinski. 2009. "Modeling Storm Impacts on Beaches, Dunes and Barrier Islands." *Coastal Engineering* 56(11): 1,133–1,152.
- Sallenger Jr., A. H. 2000. "Storm Impact Scale for Barrier Islands." *Journal of Coastal Research* 16(3): 890–895.
- Spore, N. J., and K. L. Brodie. 2017. *Collection, Processing, and Accuracy of Mobile Terrestrial Lidar Survey Data in the Coastal Environment*. Coastal and Hydraulics Laboratory Special Report. CHL-TR-17-5. Vicksburg, MS: U.S. Army Engineer Research and Development Center.
- Stockdon, H. F., A. H. Sallenger, R. A. Holman, and P. A. Howd. 2007. "A Simple Model for the Spatially Variable Coastal Response to Hurricanes." *Marine Geology* 238(1): 1–20.
- Stockdon, H. F., R. A. Holman, P. A. Howd, and A. H. Sallenger. 2006. "Empirical Parameterization of Setup, Swash, and Runup." *Coastal Engineering* 53(7): 573–588.

- Theuerkauf, E. J., and A. B. Rodriguez. 2012. "Impacts of Transect Location and Variations in Along-Beach Morphology on Measuring Volume Change." *Journal of Coastal Research* 28(3): 707–718.
- van Rooijen, A. Reniers, J. van Thiel de Vries, C. Blenkinsopp, and R. McCall. 2012. "Modeling Swash Zone Sediment Transport at Truc Vert Beach." *33rd Conference on Coastal Engineering 2012*. Edited by P. Lynett and J. McKee Smith, Vol. 4, ICCE, 3,265–3,276.
- Vousdoukas, M.I., T. Kirupakaramoorthy, H. Oumeraci, M. De La Torre, F. Wübbold, B. Wagner, and S. Schimmels. 2014. "The Role of Combined Laser Scanning and Video Techniques in Monitoring Wave-by-Wave Swash Zone Processes." *Coastal Engineering* 83: 150–165.
- Zhang, W., J. Qi, P. Wan, H. Wang, D. Xie, X. Wang, and G. Yan. 2016. "An Easy-to-Use Airborne LiDAR Data Filtering Method Based on Cloth Simulation." *Remote Sensing* 8(6): 501.

Appendix A: Lidar Specifications

Lidar Tower Specs

Dimensions

- Primary mast that the lidar is mounted to: approximately 13 m long
- Vertical support pedestal: 6.5 m tall (location of the top pulley)
- Concrete pad: $2.44 \times 2.44 \times 1.5$ m (L×W×D)
- Vertical end-of-mast cradle support: approximately 6 m off the ground (legs of support structure were welded to pipes driven 6 m into the ground in the event that the entire dune was lost, hopefully the cradle still stands)

Lidar Mount Design

- Mounting plates: 0.6×0.6 m
- Hinge mechanism: double pinned hinge point with $\frac{3}{4}$ -inch adjustable all-thread clamps
- Range of angles: $0^\circ - 45^\circ$

Description of beach environment

- Distance from dune crest to concrete pedestal anchor: 12 m
- Beach width: 20 – 40 m depending on seasonal and tide conditions
- Dune crest elevation under scanner: 6.7 m (NAVD88)
- Lightning protection: three 12 m telephone poles with lightning rods surrounding the system

VZ1000-PH specs

- Hexagonal glass tube consisting of 6 plane-parallel glass panes with anti reflective coating to minimize back reflections of light emitted by the scanner
- Dimensions: $0.86 \times 0.45 \times 0.49$ m
- Weight: 52 kilograms (kg) (without scanner)
- Protection Class: IP65
- Power Supply: 22-26V DC
- Current Consumption: 14 A @ 24 V DC (without scanner)
- Temperature Range: -20°C to $+45^\circ\text{C}$
- Power/Communication Connection: Ruggedized Harting locking connector

VZ1000 specs

- Laser Product Classification: Class 1
- Laser Wavelength: 1550 nm
- Scanning Mechanism: rotating multi-facet mirror (three facets) on rotating head
- Vertical (Line) Scan
 - Theta angle range: 100° ($+60^{\circ}$ / -40°)
 - Scan speed: 3 lines/second (s) to 120 lines/s
 - Angular step width: variable between 0.0024° to 0.288°
 - Angle measurement resolution: 0.0005°
- Horizontal (Frame) Scan
 - Phi angle range: 360 deg (0--360)
 - Scan speed: $0^{\circ}/s$ to $60^{\circ}/s$
 - Angular step width: 0.0024 to 0.5°
 - Angle measurement resolution: 0.0005°
- Measurement principle: single-shot time-of-flight measurement with echo digitization and online waveform processing
- Laser Pulse Repetition Rate, Peak (PRR): four selectable modes: 70 kilohertz (kHz), 100 kHz, 150 kHz, 300 kHz
- Maximum Measurement Range: 450 m @ 300 kHz to 1400 m @ 70 kHz
- Minimum Range: 2.5 m
- Accuracy: 8 mm
- Precision: 5 mm
- Beam Divergence: 0.3 milliradians (mrad) (30 millimeter [mm] beam width increase per 100 m of range)
- Dimensions: 0.2×0.3 m
- Weight: 9.8 kg
- Protection Class: IP64
- Temperature Range: 0°C to $+40^{\circ}\text{C}$
- Power Supply Range: 11–32 V DC
- Current Consumption: maximum 7.5 A @ 12V DC or maximum 3.7 A @ 24 V DC

Data transmission and connectivity

The data are transmitted from the scanner via CAT6 Ethernet cable down the spine of the mast (quality Ethernet data transmission is limited to 100 m cable length before passing through a switch or booster). At the base of the mast, it runs underground into the power and communications cabinet located approximately 4 m behind the vertical support pedestal. Inside the cabinet, the data passes through a gigabit Ethernet switch and exits the cabinet through another CAT6 Ethernet cable routed underground in rigid conduit. The cable runs approximately 30 m away from the cabinet to an additional data acquisition trailer that is simultaneously receiving additional oceanographic sensor data. The cable passes through another gigabit switch and a fiber converter, converting the signals from copper Ethernet to fiber-optic. The fiber runs approximately 450 m underground back to the primary data acquisition office, where it is converted back to Ethernet and the data are stored within large network attached storage arrays. Commands from the data acquisition computers located within the primary office are routed using the same fiber-Ethernet path previously described.

Appendix B: Parameters File Format

The parameters file is a MATLAB file containing a number of variable declarations. It is used to define how the DuneLidar code will perform a number of operations during the processing of data. A script called *generateParamsFile.m*, which can be used to generate a parameters file, is provided in the Setup/params/ directory of the FRF Remote Sensing DuneLidar git repository, located here: [/FRF-Remote-Sensing/DuneLidar](#).

Parameters file variables

All variables in the parameters file must be stored in the params structure (e.g., *params.RXPconvert*). The following list will omit this variable name for the sake of avoiding repetition. The variables are grouped by type only to aid in more clear documentation. Some variables are optional and do not need to be defined for the DuneLidar code to execute properly. These are indicated by *italics*.

Executables and file locations

RXPconvert	The full path to the RXPconvert executable
cloudfilters	The full path to the cloud-filters executable
points2grid	The full path to the points2grid executable
lasmerge	The full path to the lasmerge executable
duneLidarRepoLocation	The full path to the DuneLidar repository

Coregistration parameters

coreg.xyzGCPs	A 2D matrix of GCP locations, where each row is a single GCP and columns are <i>x</i> -, <i>y</i> -, and <i>z</i> -coordinates.
coreg.rmsErrorThreshold	The root mean square error threshold

Framescan parameters

frame.rxpreturns	The returns to read from RXP files
frame.rxpvalues	The data fields to read for each return from RXP files
frame.filter.intensity	The data field to use as the intensity value in the cloud-filters program
<i>frame.filter.passthrough</i>	A string containing the command-line options for the passthrough filter when using cloud-filters
<i>frame.filter.noise</i>	A string containing the command-line options for the noise filter when using cloud-filters
<i>frame.filter.range</i>	A string containing the command-line options for the range filter when using cloud-filters
<i>frame.filter.scanthreshold</i>	A string containing the command-line options for the scan threshold when using cloud-filters
frame.keep_individual_las	A boolean value indicating whether to keep or delete LAS files if they are generated when reading data from the RXP files
frame.grid	The command-line options used to define the DEM grid when using points2grid
frame.demtypes	The command-line option used to define the gridding method to use when creating the DEM with points2grid
frame.demformat	The command-line option used to define the output format of the DEM when using points2grid

Linescan parameters

runup.precision	Lidar scanner range precision; used as threshold for defining water height above ground
runup.filtseconds	Length of smoothing frame for the minimum filter, in seconds.
runup.xShoreSmooth	Meters to smooth in the cross-shore
runup.xShoreSmooth2	Meters to smooth in the cross-shore and fill in NaNs
runup.timeSmooth	Seconds to smooth in time
runup.cumNormThresh	Percentage of time above minimum foreshore to be called water
line.P1	An array containing two values, the X and Y coordinates of the point P1, which is used in defining the downline vector
line.P2	An array containing two values, the X and Y coordinates of the point P2, which is used in defining the downline vector
line.downline_vector	An array that defines locations along the downline vector that will be used in gridding. For example, -5 : 0.1 : 150 will generate an array of values from -5 to 150 with a spacing of 0.1. This corresponds to a location -5 m along the vector defined as (P1, P2) to a location 150 m along the same vector, with a grid spacing of 0.1 m.

- line.VirtualGaugeXLocations** An array of X values, corresponding to the *x*-coordinates of the virtual wave gauges (in FRF coordinates)
- line.VirtualGaugeYLocations** An array of Y values, corresponding to the *y*-coordinates of the virtual wave gauges (in FRF coordinates)
- line.VirtualGaugeLat** An array of latitude values, corresponding to the latitudes of the virtual wave gauges
- line.VirtualGaugeLon** An array of longitude values, corresponding to the longitudes of the virtual wave gauges (in FRF coordinates)

Plotting parameters

Each of the following parameters falls into the `images` struct (and so will begin with `params.images`)

- coreg.xlims** An array containing two values, the min and max limits of the *x*-axis for the coregistration plot
- coreg.ylims** An array containing two values, the minimum and maximum limits of the *y*-axis for the coregistration plot
- coreg.cMap** Custom color map for Z Difference Plot, makes zero change white
- frame.limits** An array containing four values, the minimum and maximum limits of the *x*-axis and the minimum and maximum limits of the *y*-axis for the framescan plot
- xlims** An array containing two values, the minimum and maximum limits of the *x*-axis, to be used for all plots that do not have an otherwise specified limit

ylimits	An array containing two values, the minimum and maximum limits of the <i>y</i> -axis, to be used for all plots that do not have an otherwise specified limit
inst.numToPlot	The number of scanlines to plot in the instant linescan plot
inst.numToSkip	Instead of plotting inst.numToPlot consecutive linescan, this parameter will skip a number of scanlines between the ones that are plotted (a value of two will plot scanlines 1, 4, 7, etc.)
foreshore.filtseconds	Length of smoothing frame for the filter, in seconds
foreshore.filtmeters	Width of smoothing frame for the filter, in meters
foreshore.xlimsShore	An array containing two values, the minimum and maximum limits of the <i>x</i> -axis of the foreshore section of the foreshore plot
foreshore.ylimsShore	An array containing two values, the minimum and maximum limits of the <i>y</i> -axis of the foreshore section of the foreshore plot
foreshore.caxisShore	An array containing two values, the minimum and maximum values defining the color range of the colorbar of the foreshore section of the foreshore plot
foreshore.xlimsWater	An array containing two values, the minimum and maximum limits of the <i>x</i> -axis of the water section of the foreshore plot
foreshore.ylimsWater	An array containing two values, the minimum and maximum limits of the <i>y</i> -axis of the water section of the foreshore plot

foreshore.caxisWater An array containing two values, the minimum and maximum values defining the color range of the colorbar of the water section of the foreshore plot.

runup.ylimits An array containing two values, the minimum and maximum limits of the y -axis of the runup plot.

Appendix C: RXPconvert

RXPconvert is a program, written in C++, used to convert point cloud data from the proprietary RXP format to a binary or textual format. All code for *RXPconvert* can be found in the FRF Remote Sensing git repository, located here: </FRF-Remote-Sensing/RXPconvert>

System requirements

For both Windows and Linux, the RiVLib libraries are required. These can be obtained from Riegl.

Windows

- RiVLib
- Visual Studio

Linux

- RiVLib
- CMake
- A C++ compiler, such as GCC

Compiling

Windows

Open the `RXPconvert.vcxproj` file in Visual Studio and compile.

Linux

In a terminal window, navigate to the repository directory and execute the following commands:

- `mkdir build`
- `cd build`
- `cmake .. -DCMAKE_PREFIX_PATH=/path/to/rivlib/`
- `make`

where `/path/to/rivlib/` is the full path to the RiVLib directory.

Usage

The *RXPconvert* program can read data from an RXP file and write to either a binary or text file. It is run at the command line with the following usage pattern:

```
./RXPConvert [rxp file] [output file] [returns]  
[fields]
```

where the options are defined as follows:

- **[rxp file]** – the full path to the RXP file
- **[output file]** – the full path to the output file. This file must have one of the following extensions:
 - **.txt** – Makes a comma delimited file with headers
 - **.bin** – Makes a binary file (each value is an 8-byte double)
 - **.csv/other** – Makes a comma delimited file without headers
- **[returns]** – defines which points to return, must be one of the following:
 - **allshots** – returns a point even when no point was collected by the scanner
 - **allpoints** – returns every point collected by the scanner
 - **first** – returns only the first point collected from each laser pulse
 - **last** – returns only the last point collected from each laser pulse
- **[fields]** – a list of characters which defines the data fields to be written for each return. Optional arguments are the following:
 - **a** – amplitude (dB)
 - **b** – background_radiation
 - **d** – deviation
 - **R** – echo_range
 - **f** – facet number
 - **h** – is_high_power
 - **p** – is_pps_locked
 - **e** – is_rising_edge
 - **s** – is_sw
 - **r** – reflectance (dB)

- **tgps** – GPS time (cannot be used with 'tint')
- **tint** – Internal time (cannot be used with 'tgps')
- **x** – x-position of point in the SOCS (m)
- **y** – y-position of point in the SOCS (m)
- **z** – z-position of point in the SOCS (m)
- **0** – beam_origin[0] (x-distance from phase center) (m)
- **1** – beam_origin[1] (y-distance from phase center) (m)
- **2** – beam_origin[2] (z-distance from phase center) (m)
- **3** – beam_direction[0] (x-component of unit direction vector)
- **4** – beam_direction[1] (y-component of unit direction vector)
- **5** – beam_direction[2] (z-component of unit direction vector)
- **n** – number of return
- **N** – number of returns for given pulse

Appendix D: 3ddconvert

3ddconvert is a program, written in C++, used to convert point cloud data from the proprietary 3dd format to binary or text format. All code for the *3ddconvert* program can be found in the FRF Remote Sensing git repository, located here: </FRF-Remote-Sensing/3ddconvert>.

System Requirements

3ddconvert can only be compiled and used on a Windows operating system due to compatibility issues with the libraries provided by Riegl. For *3ddconvert* to work, the RiScanLib library must be installed with a valid license. Additionally, the boost and liblas libraries are required if building with LAS file output compatibility. Links to these libraries are located in the README file. By default, LAS file compatibility is commented out of the code and these libraries are not needed.

Compiling

Without LAS file compatibility

Use Visual Studio to open the project files, located in the vs2012 directory of the project repository. Build the solution to generate the *3ddconvert* executable.

With LAS file compatibility

If LAS file compatibility is required, the boost and liblas libraries must first be installed. After installing the boost and liblas libraries, use Visual Studio to open the project files, located in the vs2012 directory of the project repository. Uncomment lines 95–164 of *3ddconvert.cpp* and comment out line 93.

Ensure that the compiler has access to the header files for all libraries and that the linker file has access to all dll files by doing the following:

Go to Project > 3ddconvert Properties and set the following:

- Configuration Properties
 - C/C++
 - General

- Additional Include Directories
 - liblas-master/include or libLAS-1.8.0
 - boost_1_61_0
 - Code Generation
 - Runtime Library
 - Multi-threaded DLL (/MD)
 - Linker
 - General
 - Additional Library Directories
 - liblas-master/build/bin/Release or libLAS-1.8.0/build/bin/Release
 - boost_1_61_0/stage/lib
 - Input
 - Liblas.lib

Ensuring that the paths used are the ones corresponding to your installed library versions.

Finally, build the solution to generate the *3ddconvert* executable.

Usage

The *3ddconvert* program can read data from a 3dd file and write to either a binary or text file (or LAS file if compiled with compatibility). It is run at the command line with the following usage pattern:

```
3ddconvert.exe [input file] [output file]
```

where the options are defined as follows:

- [input file] – the full path to the 3dd file
- [output file] – the full path to the output file. This file must have one of the following extensions:
 - .txt – Makes a comma delimited file with headers
 - .bin – Makes a binary file (each value is an 8-byte double)
 - .csv – Makes a comma delimited file without headers
 - .las – Makes an LAS file

Because the 3dd format is older and less robust than the RXP format, fewer data fields are available for each return. Therefore, the *3ddconvert* program always outputs every field for every return. These include *x*-, *y*-, and *z*-coordinates in the scanner's own coordinate system, as well as return intensity and return time.

Appendix E: readLidar

The *readLidar* script is a *MATLAB* script used to read point cloud data from either 3dd or RXP format into *MATLAB*. All code for the *readLidar.m* script can be found in the lidar directory of the FRF Remote Sensing Utilities git repository, located here: [/FRF-Remote-Sensing/Utilities](#)

Usage

The *readLidar.m* script uses the *varargin* and *varargout* features of *MATLAB* to accept and return multiple optional arguments. The arguments passed to the script will govern how the data are read from file.

Required Inputs

The following are required inputs for the *readLidar.m* script:

filenames A string or a cell array of strings of paths to rxp or 3dd file(s), or zip files that contain rxp or 3dd files. If a zip file is passed, the script will automatically unzip, read the data, and delete the unzipped files, leaving the zip file unchanged.

Optional inputs

The following inputs are optional, and will govern the way that data are read from the provided files. Optional inputs are provided as key, value pairs of strings, where the key indicates the option and the value indicates the actual optional value.

returns Which returns are read from the file. This value is only used when reading from rxp files, as by default, *3ddconvert* reads all returns.

Possible values: allshots, allpoints, first, last

Default value: allpoints

types A cell array containing the data fields to read for each return. Certain values are only available from the rxp format, so refer to the RXPconvert documentation (Appendix C) for the full list.

Default value: {'x', 'y', 'z', 'r', 'tgps'}

The following additional fields are available to define how GPS timing data are handled:

tgps Always tries to fill in gaps with GPS data. Will use filename datetime if necessary.

tgps_datenum Same as **tgps** but will convert values to *MATLAB* datenums using a datetime string found in the filename.

tint The lidar scanner's own internal time.

tfilename Generate GPS time using a datetime string found in the filename and the lidar scanner's internal time.

tgps_interp Interpolates the gaps between sections of good GPS timing data. If the entire file contains no GPS data, will return all NaNs.

tgps_interp_datenum Same as **tgps_interp** but will convert values to *MATLAB* datenums using a datetime string found in the filename.

tgps_raw The raw GPS timing data, without trying to fill any gaps in data.

tgps_raw_datenum Same as **tgps_raw** but will convert values to *MATLAB* datenums using a datetime string found in the filename.

exe The full path to the RXPconvert executable.
Default: 'RXPconvert.exe'

Tdd The full path to the 3ddconvert executable.

Default: '3ddconvert.exe'
dbg Print debugging information while the script is running. To enable debugging output, pass the string 'yes' to this option.

rmat A rotation matrix to apply to the position data, once it has been read. This assumes that 'x', 'y', and 'z' are the first three arguments of the types cell array.

Default: eye(4)

Outputs

The *readLidar* script returns a matrix containing the requested data. The size of the matrix, mxn , will be determined by two factors. The matrix will contain m rows, where m is the number of data fields read for each point. Therefore, the number of rows will be equal to the number of arguments passed to the **types** option. The matrix will contain n columns, where n is

the total number of returns read from all files. For example, if the script is passed two rpx files, each containing 1000 points, and is instructed to read all points, the matrix will contain 2000 columns.

Additionally, the *readLidar* script returns a $1 \times n$ array, which contains, for each return, the scanline number that that point belongs to. Returns are considered part of the same scanline if they are collected from the same mirror face in the lidar scanner. Starting with 1, the scanline number is incremented by one each time the next mirror face begins collection.

Examples

The following examples show how to call the *readLidar.m* script.

Read data from file using all defaults for optional arguments:

```
readLidar('/path/to/file.rpx')
```

Read only the x, y, and z coordinates from the file:

```
readLidar('file.rpx', 'types', {'x', 'y', 'z'})
```

Read the data from two 3dd files, and specify the path to 3ddconvert:

```
readLidar({'file1.3dd', 'file2.3dd'}, 'exe', '/path/to/3ddconvert')
```

Read all returns, including only point coordinates:

```
readLidar('file.rpx', 'returns', 'all', 'types', {'x', 'y', 'z'})
```

Appendix F: Cloud-Filters

The cloud-filters program is a command-line tool that can be used for the filtering and analysis of point cloud data. It is written in C++ and the full documentation and code are available in the FRF Remote Sensing git repository at: [/FRF-Remote-Sensing/cloud-filters](#)

System requirements

Currently, only the Linux operating system is supported. However, building for Windows should be possible as all required libraries are supported on Windows.

- **PCL** – The Point Cloud Library, which is used for visualization and point cloud Input/Output operations
- **CMake** – Used for building makefiles
- **GNU Make** – Used to compile the code into an executable
- **A C++ compiler such as gcc or g++**
- **gnuplot** (optional) – Used to create plots

Compiling

Compiling on Linux is fairly simple. Once you have installed all required libraries, run the following commands:

- `cd /path/to/cloud-filters`
- `mkdir build`
- `cd build`
- `cmake ..`
- `make`

Once the code has finished compiling, the cloud-filters executable will be located in the `/path/to/cloud-filters/build` directory.

Usage

The cloud-filters program works by executing filters and tools in the order that they appear at the command-line. Therefore, operations can be strung together in a sequence to perform useful tasks such as filtering data, generating images, and converting file formats. The following command-line arguments are the exception to the rule, and can be placed in any

location within the command without having an effect on the execution order of the filters and tools:

- **--input** – Sets the input point cloud
- **--quiet** – Turns off verbose output
- **--transects** – Sets the data file that contains transect definitions
- **--scanthreshold** – Sets the angle to be used when extracting scanlines
- **--scanner** – Sets the scanner location
- **--output** – Sets the output file(s)

Input/Output flags

The following flags are used for input/output operations. They define how files will be read and how data will be written to file.

--input

Defines the input file to be used as the raw, unfiltered point cloud. Two different formats are accepted, .pcd and a custom binary format defined below.

Usage

```
_____--input [filename]  
  
--input [filename] [field]
```

Options

- **[filename]** – Full or relative path to a point cloud file
- **[field]** (conditional) – A character matching one of the characters in the file extension, whose value will be stored as the intensity value for each point

Details

The input file can be one of two formats:

- **PCD Format** – Described in the PCL documentation, the point cloud format used in PCL. The point-clouds program uses PointXYZI data types for all point clouds. For reference on point types, refer to the PCL documentation.

- **Binary Format** – This is a custom binary format, used to read data saved in *MATLAB* using the `fwrite()` function. Saved data must be an $m \times n$ matrix, where m is the number of fields per point (e.g., three if only x, y, and z are saved) and n is the number of points in the point cloud. The length of the file extension must be equal to the value of m . For example, a file containing x, y, z, amplitude, and reflectance values for each point could have a .xyzar file extension. When this type of file is used, a second flag must be included to indicate which value in the binary data will be stored as the intensity value for each point. This flag must match one of the characters in the file extension. In the previous example, using the 'r' character as the second flag will store the reflectance value of each point as the intensity value in the cloud-filters program. Note that x, y, and z values must always be included, and they must always be the first three values for each point.

Examples

Read the `cloud.pcd` file:

```
./cloud-filters --input ../../data/cloud.pcd
```

Read the binary file `cloud.xyzar` and store amplitude as intensity:

```
./cloud-filters -input ../../data/cloud.xyzar a
```

–output

Defines the output file(s) used to save ground and optionally non-ground points.

Usage

```
--output [ground points file]
```

```
--output [ground points file] [non-ground points file]
```

```
--output [bool file]
```

Options

- **[ground points file]** – The full or relative path to the file that will contain ground points

- **[non-ground points file]** (optional) – The full or relative path to the file that will contain non-ground points
- **[bool file]** – The full or relative path to the file that will contain an array of ground/non-ground boolean values

Details

Output file format will be determined by the extension given to the filename. A mix of file types can be used (i.e., ground file in pcd format and non-ground in xyz format). The following extension can be used:

- **pcd** – PCD format, described here
- **xyz** – A binary format containing only *x*-, *y*-, and *z*-coordinates, as densely packed floats
- **bool** – This format is a single file that contains ground and non-ground points. It saves a binary file in which each bit represents a boolean value indicating whether the point at the index of that bit is ground or non-ground. The data is padded to 8 bits at the end of the file if necessary.
- **las** – This format is a single file that contains ground and non-ground points. See here for documentation. Points are classified using the ASPRS Standard Lidar Point Classes, where ground points are given a classification value of 2 (Ground) and non-ground are given a classification value of 1 (Unclassified). A scale factor of 0.0001 is used.

Examples

Read `cloud.pcd` and save it as `samecloud.pcd`:

```
./cloud-filters --input cloud.pcd -output  
samecloud.pcd
```

Read `cloud.xyzar` and store reflectance as amplitude, and then save to `cloud.pcd`:

```
./cloud-filters --input cloud.xyzar a --output  
cloud.pcd
```

Read `cloud.pcd`, save ground points to `samecloud.pcd` and non-ground points to `nocloud.pcd` (which will not be created since there will be no non-ground points):

```
./cloud-filters --input cloud.pcd --output  
samecloud.pcd nocloud.pcd
```

Filters

The following flags can be used to filter point clouds by separating them into ground and non-ground points.

--passthrough

Simple filter used to include or exclude points inside or outside of a given range of values. Range can be defined in x, y, z, or intensity fields.

Usage

```
--passthrough [include/exclude] [x/y/z/intensity] [min] [max]
```

Options

- **[include/exclude]** – Include will consider points inside of the given range ground points and points outside of the given range non-ground points. Exclude will consider points inside of the given range non-ground points and points outside of the given range ground points.
- **[x/y/z/intensity]** – The axis or field to which the filter will be applied.
- **[min]** – The lower bound of the range (inclusive)
- **[max]** – The upper bound of the range (inclusive)

Examples

Extract points in the x-range [1.5, 5.5] from `cloud.pcd` and save to `smallcloud.pcd`:

```
./cloud-filters --input cloud.pcd --passthrough  
include x 1.5 5.5 --output smallcloud.pcd
```

--range

A filter that separates ground points from non-ground points based on the distance of consecutive points within a scan line from the scanner itself.

Usage

--range [direction] [stencil size] [stencil pass] [distance function]

--range [direction] [stencil size] [stencil pass] [distance function]
[turning point method]

--range [direction] [stencil size] [stencil pass] [distance function]
[turning point method] [turn skip]

Options

- **[direction]** – Must be either “up” or “down.” Describes the direction the scanner performed individual line scans (i.e., up means it scanned low to high)
- **[stencil size]** – The number of adjacent points to consider when deciding if a point is ground or non-ground
- **[stencil pass]** – The number of points inside the stencil that the current point must properly align with to be considered a ground point
- **[distance function]** – The distance function to use when determining the distance between a point and the scanner. Must be one of “ground,” which calculates distance in the x - y plane, or “air,” which calculates distance in 3D space.
- **[turning point method]** – Include this option if scanlines cross the y -plane of the scanner. Two options are available for determining the point at which the distance of consecutive points from the scanner should switch from closer to further (or vice versa...this is called the turning point). The “theta” method defines the turning point as the point at which the change in angle between the scan and the horizon switches from positive to negative (or vice versa). The “closest” flag uses the point closest to the scanner (using the ground distance function) as the turning point.
- **[turn skip]** – The number of points to skip in a linescan before looking for the turning point. Useful when a portion of the first returns in a scan are air noise right next to the scanner. Default is 0.

Details

The range filter is described in detail in appendix H.

Examples

Filter `cloud.pcd` using the simplest range filter and save ground points to `filtered.pcd`:

```
./cloud-filters --input cloud.pcd --range down 1
1 --output filtered.pcd
```

Filter `cloud.pcd` using a range filter with a stencil size of 8 and a stencil pass of 5, and save ground and non-ground points to the files `ground.pcd` and `nground.pcd` respectively:

```
./cloud-filters --input cloud.pcd --range down 8
5 --output ground.pcd nground.pcd
```

Point cloud statistics

The following flags can be used to print statistics about the point cloud at any point during processing.

--bounds

Prints the bounding box of the current set of ground returns.

Examples

Print bounds of `cloud.pcd`:

```
./cloud-filters --input /home/data/cloud.pcd
--bounds
```

Print bounds of `cloud.pcd` after removing intensity values less than 0.0:

```
./cloud-filters --input /home/data/cloud.pcd
--passthrough include intensity 0 99999 --bounds
```

Scanline highlighting

This tool allows the user to "highlight" individual scanlines by setting the intensity of the points of each scanline to a very large value. The `pcl_viewer` program can then be used to display the highlights by coloring by intensity.

--highlight

Highlights scans at a set interval.

Usage

--highlight [interval]

Options

- **[interval]** – Defines the interval at which scans will be highlighted. For example, 2 will highlight every other scan, and 10 will highlight every 10th scan.

Details

The value of [interval] must be greater than or equal to 1 and will default to 2 if invalid input is provided.

Examples

Highlight every 10th scan:

```
./cloud-filters --input /home/data/cloud.pcd  
--highlight 10 --output /home/data/cloudh.pcd
```

Transects

Various types of transect data and images can be produced at any point during the filtering process. Two flags are required for generating transects. The first defines a list of transects that are to be created, and the second, which can be used multiple times in an execution, triggers the creation of a new set of transects. Note that transect generation is only supported on linux systems with gnuplot installed.

--transects*Usage*

--transects [transect file]

Options

- **[transect file]** – Text file containing any number of transect definitions

Details

This flag is required when using the `--transect` flag to toggle the creation of transect data/images. It can appear anywhere in the command-line inputs. See transect definitions for details on how to define various types of transects.

Examples

Create transects defined in `transects.txt` without any filtering:

```
./cloud-filters --input /home/data/cloud.pcd --  
transect --transects /home/data/transects.txt
```

--transect

Toggles the creation of all transects in the file defined using the `--transects` flag.

Details

This flag is part of the ordered execution of the command-line options. It can be used multiple times to create a series of data/images, which will be numbered based on their order in the command-line inputs.

Examples

Create transects defined in `transects.txt` before and after removing points outside of the range `[10, 100]`:

```
./cloud-filters --input /home/data/cloud.pcd  
--transect --passthrough include x 10 100 --transect  
--transects /home/data/transects.txt
```

Transect definitions

The following transect types are available and defined as follows. Transect definitions are kept in a single text file, with one transect per line.

Profile

Plots a profile view along x -, y -, or z -axis.

Usage

```
profile [output directory] [basename] [axis] [center] [padding]
[xmin] [xmax] [ymin] [ymax] [clean]
```

Options

- **[output directory]** – The top level directory to store profile images
- **[basename]** – The basename for output files
- **[axis]** – The plane axis which will be flattened to create the profile view (x , y , or z)
- **[center]** – The value which defines the plane in the $[axis]$
- **[padding]** – The distance in either direction of $[center]$ in which to include points in the plot
- **[xmin]** – Minimum value of the plot's x -axis
- **[xmax]** – Maximum value of the plot's x -axis
- **[ymin]** – Minimum value of the plot's y -axis
- **[ymax]** – Maximum value of the plot's y -axis
- **[clean]** – (optional) Including the word “clean” will remove data and command files used to generate the plot.

Details

Setting the bounds of either the x -range or y -range equal to each other will cause the plot to use the bounds of the plotted data as that axis's range.

Data files will be stored in the `/data/` subdirectory of $[output\ directory]$ and command files used to create images will be stored in the `/cmd/` subdirectory. Only ground points are plotted.

Examples

Sample content of a file called `transects.txt`:

```
profile /home/transects profile1 y 1000 5 45 70 -
2 10 clean
```

```
profile /home/transects profile2 y 1010 5 45 70 -
2 10 clean
```

Scanline

Plots a single scanline.

Usage

```
scan [output directory] [basename] [axis] [scan] [xmin] [xmax]  
[ymin] [ymax] [clean]
```

Options

- **[output directory]** – The top level directory to store profile images
- **[basename]** – The basename for output files
- **[axis]** – Must be one of the following characters:
 - **x** – The scan as viewed looking down the *x*-axis
 - **y** – The scan as viewed looking down the *y*-axis
 - **z** – The scan as viewed looking down the *z*-axis
 - **g** – The *x*-axis of the plot will be ground distance from the scanner, *y*-axis will be elevation
 - **a** – The *x*-axis of the plot will be air distance from the scanner, *y*-axis will be elevation
 - **n** – The *x*-axis of the plot will be the index in the scan, *y*-axis will be elevation
- **[scan]** – The scan index (0 is the first scanline)
- **[xmin]** – Minimum value of the plot's *x*-axis
- **[xmax]** – Maximum value of the plot's *x*-axis
- **[ymin]** – Minimum value of the plot's *y*-axis
- **[ymax]** – Maximum value of the plot's *y*-axis
- **[clean]** – (optional) Including the word “clean” will remove data and command files used to generate the plot.

Details

Only ground points are plotted. Setting the bounds of either the *x*-range or *y*-range equal to each other will cause the plot to use the bounds of the plotted data as that axis's range.

Examples

Sample content of a file called `transects.txt`:

```
scan /home/transects scan0 g 0 clean
```

```
scan /home/transects scan100 n 100 clean
```

Appendix G: *points2grid*

The *points2grid* program generates DEM using a local gridding method. The local gridding algorithm computes grid cell elevation using a circular neighborhood defined around each grid cell based on a radius provided by the user. This neighborhood is referred to as a bin while the grid cell is referred to as a DEM node. Several values, including minimum, maximum, mean, and inverse distance weighted (IDW) mean, are computed for points that fall within the bin. These values are then assigned to the corresponding DEM node and used to represent the elevation variation over the neighborhood represented by the bin. If no points are found within a given bin, the DEM node receives a value of null. *points2grid* also provides a null filling option, which applies an inverse distance weighted focal mean via a square moving window of 3, 5, or 7 pixels to fill cells in the DEM that have null values.

Numerous versions of *points2grid* can be found on the internet, as the program has a fairly lengthy history. The version used at the FRF, and the most up-to-date and actively developed version of *points2grid*, is maintained by CRREL and can be found in their Github repository at <https://github.com/CRREL/points2grid>. Additional information about the *points2grid* program can be found on its OpenTopography page, located here: <http://www.opentopography.org/otsoftware/points2grid>.

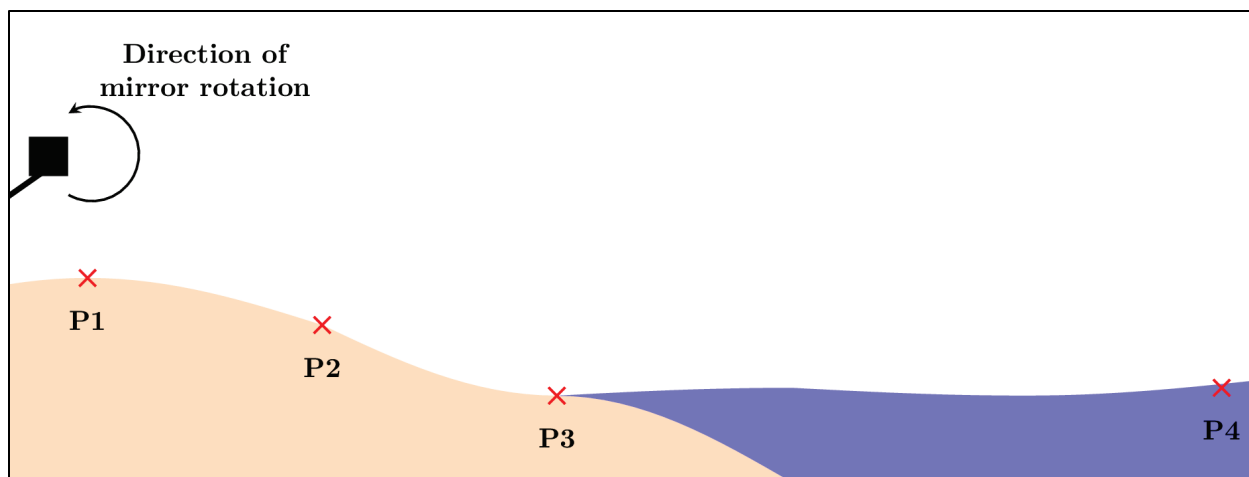
Appendix H: Range Filter

The range filter described in this appendix is used to remove noise from a point cloud dataset. The authors consider “good returns” to be those that come from the ground or water surface and “bad returns” to be those that come from objects in the air, such as birds, sand particles, or vegetation. The positioning of the lidar system within the environment at the FRF allows a simple assumption about how the data should be organized as it is collected by the scanner. Using this assumption, this filter attempts to remove the bad returns from the linescan datasets. The *MATLAB* code for this filter can be found in the *Linescan/PreProcessing/filterRaw/* directory of the FRF Remote Sensing DuneLidar git repository, located here: [/FRF-Remote-Sensing/DuneLidar](#). Additionally, the filter is implemented in C++ in the cloud-filters program, which can be found in Appendix F.

Theory

The range filter operates under the assumption that if the lidar scanner is performing a low-to-high scan, the distance of consecutive returns from the scanner should generally be increasing. This filter works because of the current collection parameters of the VZ-1000 at the FRF, which dictates the scan direction of the data collection. To illustrate this idea, refer to Figure H-1.

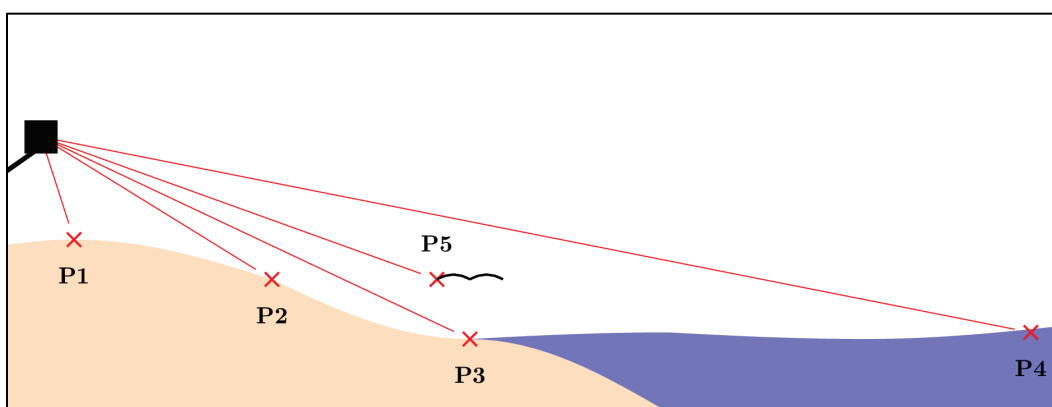
Figure H-1. Lidar scanner rotation against cross-section of the beach environment.



As can be seen in Figure H-1, because the environment of interest is generally flat and expands out to the horizon, when the lidar scanner's internal mirror is rotating counterclockwise (as viewed from the cross-section in Figure H-1), consecutive points are farther from the scanner. In this scenario, P1 would be collected first, followed by P2, P3, and then P4.

If an object that is desired to be filtered out is introduced into the environment, a bird for example, this ordering is interrupted, as illustrated in Figure H-2.

Figure H-2. A bird introduces bad returns into the point cloud.



With the bird in the environment, the points are now collected in the following order: P1, P2, P3, P5, P4.

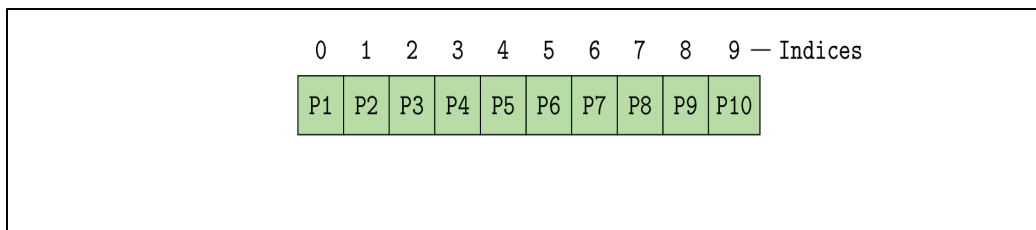
Therefore, the filter itself operates by testing consecutive returns within a single linescan. For each point, it calculates the horizontal distance of that point from the center of the scanner. If that distance is less than that of the previous point, then the point needs to be filtered out.

Implementation

In practice it has been found that because lidar scanners are generally very high resolution and consecutive returns are physically very close to each other, comparing each return only to its immediate predecessor is often too strict of a test. Therefore, the filter itself uses a moving stencil to perform the tests, allowing the user to fine-tune the filter to their environment.

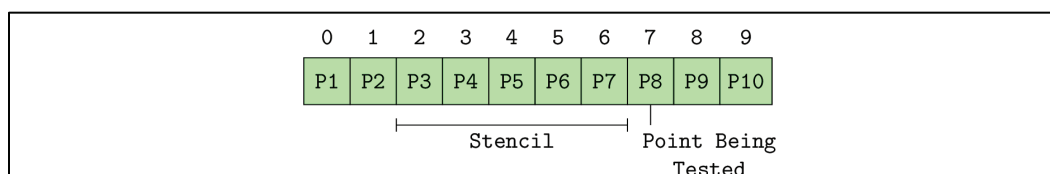
After it is collected, linescan data are stored in a 1D array, with the first entry into the array corresponding to the first return collected, the second entry corresponding to the second return, and so on, as shown in Figure H-3.

Figure H-3. Example moving stencil for filter. The first 10 returns stored in an array.



The user defines a value, called `stencilSize`, which defines the size of the stencil. This value is the number of preceding values in the array that will be tested against when looking at each return. Therefore, if the stencil size is 5, when the filter is looking at P8, the previous 5 points in the array, P3 through P7, will be considered, as shown in Figure H-4.

Figure H-4. Five point stencil with points P3 through P7 are considered when testing point P8.



The user also defines a value, called `stencilPass`, which defines the number of points within the stencil that the current point must test positively against in order for the point to be considered a ‘good’ point. In this example, if the value of `stencilPass` is 2, then P8 must be further from than scanner that at least two of the points in the stencil (i.e. points P3 through P7) for it to be considered a ‘good’ point.

Once the test is complete and the filter has decided if a point is good or bad, the filter advances to the next point. The stencil will also be advanced, but only if the point that has just been tested is considered a good point. In this example, when moving on to test P9, P3 will be removed from the beginning of the stencil and P8 will be added to the end, but only if P8 is a good point. Otherwise, the stencil remains unchanged. Continuing in this pattern, if P8 is a bad point but P9 is a good one, when the filter moves on to test P10, P9 will be placed into the stencil, which will then look as follows: [P4, P5, P6, P7, P9].

The *MATLAB* implementation of the filter has the following signature:

```
function [rfilt, zfilt] = RangeFilter(rmat, zmat, stencilSize,  
stencilPass)
```

With the function arguments defined as follows:

- **rmat** – The `rmat` variable, which is a 2D matrix of range values for every lidar return in the dataset. The range value is the horizontal distance of the return from the scanner
- **zmat** – The `zmat` variable, which is a 2D matrix of elevation values for every lidar return in the dataset. These values are not used in the actual filtering. Rather, a copy of this matrix with bad returns removed will be returned. This does not necessarily need to be the `zmat` matrix, but can be any matrix with the same dimensions as `rmat`.
- **stencilSize** – The size of the stencil, as described above.
- **stencilPass** – The number of points within the stencil that the current point must be positively tested against to be considered a good point.

And the return values defined as follows:

- **rfilt** – A filtered copy of the `rmat` matrix. Points that have been removed by the filter will be replaced with NaN.
- **zfilt** – A filtered copy of the `zmat` matrix. Points that have been removed by the filter will be replaced with NaN.

Appendix I: Variable List

Table I-1. List of variables involved in lidar tower data collection and processing.

Variable	Description
aGrid	An $m \times n$ matrix of amplitude values. Cells that contain no points are filled with NaN.
aGridInterp	An $m \times n$ matrix of amplitude values with small gaps of NaN cells filled with linearly interpolated values. The gaps that are filled match the gaps that were filled when creating the zGridInterp variable.
allDataFolder	The path to the directory on the storage hard drive that receives the raw data from the collection computer.
Amat	amplitude of return pulse.
as	The alongshore Y vector defining the DEM grid, which spans from $y = 399.975$ m to $y = 1199.975$ m with 0.1 m spacing.
baselineFolder	Path to the directory containing baseline scans.
baselineFolder	The path to the directory that contains raw baseline scans.
baselineFolder	Path to the directory containing baseline scans.
bytes	The size of the .rxp files
data	The final bare earth DEM, generated after the filtered point cloud is interpolated onto a regular 0.1 m grid.
dataFolder	The path to the raw data directory.
dataFolder	Path to the directory containing raw data.
dataFolder	The path to the raw data directory.
dataFolder	Path to the directory containing raw data.
dataFolder	The path to the raw data directory.
date	the time and date these files were imported, in Day-Month-Year Hr:Min:Sec format.
datenum	The date number of time the files were imported.
directory	A directory containing one or more framescan files.
directory	A directory containing one or more framescan files.
downlineDistStd	For each grid cell, the standard deviation of the downline distance of every point that falls in the cell.
downLineIndex	An $m \times 1$ array of the down-line index of the runup position for each scan line.
downLinePosition	An $m \times 1$ array of the down-line distance of the runup position for each scan line.
downLinePosition	Location in the cross-shore profile at 10 cm increments from the lidar.

downLineX	For each grid cell, the mean x -value of every point that falls in the cell.
downLineXstd	For each grid cell, the standard deviation of the x -values of every point that falls in the cell.
downLineY	For each grid cell, the mean y -value of every point that falls in the cell.
downLineYstd	For each grid cell, the standard deviation of the y -values of every point that falls in the cell.
exe	The path to the RXPconvert executable.
exe	The path to the RXPconvert executable.
fGrid	Gridded dataset. Wave runup data with interpolated foreshore subtracted out.
frameGriddedData	Structure containing specifications and data on filtered lidar bare-earth DEM.
frameGriddedData.data	Elevation at grid cell.
frameGriddedData.info	Contains grid bounds, limits, and precision.
frameGriddedData.info.cellsizeEW	Size of cross-shore grid cell, meters.
frameGriddedData.info.cellsizeNS	Size of alongshore grid cell, meters.
frameGriddedData.info.east	Easternmost (maximum) FRF cross-shore coordinate.
frameGriddedData.info.ncols	Number of columns in data matrices, corresponding to the number of grid cells in cross-shore.
frameGriddedData.info.north	Northernmost (maximum) FRF alongshore coordinate.
frameGriddedData.info.nrows	Number of rows in data matrices, corresponding to number of grid cells in alongshore direction.
frameGriddedData.info.south	Southernmost (minimum) FRF alongshore coordinate.
frameGriddedData.info.west	Westernmost (minimum) FRF cross-shore coordinate.
frameGriddedData.reflectanceData	Reflectance at grid cell.
frameGriddedData.waterLine	x -coordinate of the location of the water line for each lidar phi angle.
frameGriddedData.waterLineStd	Standard deviation of waterline cross-shore locations.
frameGriddedData.xs	Cross-shore FRF coordinate for each cell in gridded row.
frameGriddedData.ys	Alongshore FRF coordinate for each cell in gridded column.
frameMorphology	*Proposed, not implemented.
frameMorphology.aspect	*Proposed, not implemented.
frameMorphology.aspect.aspectGrid	*Proposed, not implemented.
frameMorphology.curvature	*Proposed, not implemented.
frameMorphology.curvature.2dCurvatureGrid	*Proposed, not implemented.

frameMorphology.curvature.xCurvatureGrid	*Proposed, not implemented.
frameMorphology.curvature.yCurvatureGrid	*Proposed, not implemented.
frameMorphology.duneToe	*Proposed, not implemented.
frameMorphology.duneToe.x	*Proposed, not implemented.
frameMorphology.duneToe.y	*Proposed, not implemented.
frameMorphology.duneToe.z	*Proposed, not implemented.
frameMorphology.slope	*Proposed, not implemented.
frameMorphology.slope.2dslopeGrid	*Proposed, not implemented.
frameMorphology.slope.xSlopeGrid	*Proposed, not implemented.
frameMorphology.slope.ySlopeGrid	*Proposed, not implemented.
frameRawRectifiedData	Structure containing framescan RXP file information and rectification, coregistration matrix.
frameRawRectifiedData.filenames	Structure containing filenames and information on framescan RXP files from scan.
frameRawRectifiedData.rotMatrix	Both rectification and coregistration rotation matrix.
griddedDataIsWater	An $m \times n$ masking matrix indicating whether each point in the lineGriddedData matrices are water (i.e., 1 means the point is grid cell is water; 0 means it is not).
Hz	The average full scan line record frequency, in hertz.
info	Additional information about the grid, including the north, south, east, and west extent of the grid, the number of columns and rows in the grid, and the grid resolution.
isNoise	Contains a boolean value indicating whether each point is noise or not (i.e., 1 means the point is noise; 0 mean it is not).
isQAQC	A numeric flag indicating whether the data has been quality controlled. Automatically set to zero for real-time data.
last_processed	Will contain the collection date of the last real-time dataset that was processed. If none have been processed yet, it will contain the date that the MasterProcess.m script was started.
lidarLocation	The lidar's center in rectified coordinates.
lineCoredat	<i>MATLAB</i> structure containing core variables used to make the gridded data.
lineCoredat.downLineDistance	1D array. The distance in meters of each cell in the cross-shore direction from the point P1. Is the the average of all points that are in the cell or just the center of the cell?

<code>lineCoredat.downlineDistStd</code>	1D array. The standard deviation, in meters along the downline vector, of all LIDAR points that fall into the cell.
<code>lineCoredat.downLineX</code>	1D array. The mean X value of all LIDAR returns that fall into each cell.
<code>lineCoredat.downLineXstd</code>	1D array. The standard deviation of all X values that fall into each cell.
<code>lineCoredat.downLineY</code>	1D array. The mean Y value of all LIDAR returns that fall into each cell.
<code>lineCoredat.downLineYstd</code>	1D array. The standard deviation of all Y values that fall into each cell.
<code>lineCoredat.lineFit</code>	<i>MATLAB</i> structure containing data that defines the downline vector.
<code>lineCoredat.lineFit.P1</code>	The origin point of the downline vector.
<code>lineCoredat.lineFit.P2</code>	The point that defines the direction of the downline vector.
<code>lineCoredat.offlineDist</code>	1D array. The average distance, in meters cross-shore from the downline vector, of all lidar points that fall into the cell.
<code>lineCoredat.offlineDistStd</code>	1D array. The standard deviation, in meters cross-shore from the downline vector, of all lidar points that fall into the cell.
<code>lineCoredat.tGPSVector</code>	1D array. The average time for each scan number.
<code>lineCoredat.tGPSVectorRange</code>	<i>MATLAB</i> structure containing minimum and maximum time for each scan number.
<code>lineCoredat.tGPSVectorRange.max</code>	1D array. The maximum time for each scan number.
<code>lineCoredat.tGPSVectorRange.min</code>	1D array. The minimum time for each scan number.
<code>lineGriddedData</code>	<i>MATLAB</i> structure containing the gridded, rectified, noise-filtered data.
<code>lineGriddedData.aGrid</code>	Gridded dataset. The amplitude data.
<code>lineGriddedData.aGridInterp</code>	Gridded dataset. The amplitude data with gaps in data interpolated.
<code>lineGriddedData.zGrid</code>	Gridded dataset. The elevation data.
<code>lineGriddedData.zGridInterp</code>	Gridded dataset. The elevation data with gaps in data interpolated.
<code>lineGriddedFilteredData</code>	<i>MATLAB</i> structure containing water and foreshore grids filtered based on the runup line.
<code>lineGriddedFilteredData.foreshoreGridFiltered</code>	Gridded dataset. The filtered foreshore elevation grid.
<code>lineGriddedFilteredData.waterGridFiltered</code>	Gridded dataset. The filtered water elevation grid.
<code>lineHydrodynamicsData</code>	Structure. The hydrodynamic data are crossshore locations specified in <code>lineCoreDat</code> .

<code>lineHydrodynamicsData.asym</code>	1D array. Asymmetry of sea-surface elevation time-series.
<code>lineHydrodynamicsData.FFT</code>	Structure. Data from FFT of sea-surface elevation time-series at each cross-shore location.
<code>lineHydrodynamicsData.FFT.amp</code>	2D array. Energy in each wave frequency band at each cross-shore location.
<code>lineHydrodynamicsData.FFT.freq</code>	1D array. Center of wave frequency bands for wave energy spectra.
<code>lineHydrodynamicsData.FFT.Hsig</code>	1D array. Significant infragravity wave heights ($f < 0.05$ Hz).
<code>lineHydrodynamicsData.FFT.Hsin</code>	1D array. Significant sea/swell wave heights ($0.05 < f < 0.03$ Hz).
<code>lineHydrodynamicsData.FFT.TpIG</code>	1D array. Peak period infragravity wave waves ($f < 0.05$ Hz).
<code>lineHydrodynamicsData.FFT.TpIN</code>	1D array. Peak period sea/swell wave heights ($0.05 < f < 0.03$ Hz).
<code>lineHydrodynamicsData.Hsig</code>	1D array. Total significant wave heights of sea-surface elevation time-series.
<code>lineHydrodynamicsData.maxWaterLevel</code>	1D array. Maximum of sea-surface elevation time-series.
<code>lineHydrodynamicsData.meanWaterLevel</code>	1D array. Mean of sea-surface elevation time-series.
<code>lineHydrodynamicsData.minWaterLevel</code>	1D array. Minimum of sea-surface elevation time-series.
<code>lineHydrodynamicsData.skew</code>	1D array. Skewness of sea-surface elevation time-series.
<code>lineHydrodynamicsData.stdWaterLevel</code>	1D array. Standard deviation of sea-surface elevation time-series.
<code>lineMorphologyData</code>	Structure. Sub-aerial beach profile data and statistics from 30-minute linescan at cross-shore locations specified in <code>lineCoreDat</code> .
<code>lineMorphologyData.curvature</code>	1D Array. 1D curvature of sub-aerial beach profile.
<code>lineMorphologyData.curvature₂</code>	1D Array. Interpolated one dimensional curvature of sub-aerial beach profile.
<code>lineMorphologyData.foreshoreSlope</code>	Linear-fit of the beach profile slope in the swash zone (defined as mean water level $\pm 2 \times \text{sigma}$).
<code>lineMorphologyData.foreshoreSlopeRsqr</code>	Ordinary R-squared value for linear-fit of the beach profile slope in the swash zone.
<code>lineMorphologyData.instantaneousSlope</code>	1D Array. Slope at cross-shore locations along sub-aerial beach profile.
<code>lineMorphologyData.maxForeshore</code>	1D Array. Maximum sub-aerial beach elevation.
<code>lineMorphologyData.meanForeshore</code>	1D Array. Mean sub-aerial beach elevation.

lineMorphologyData.minForeshore	1D Array. Minimum sub-aerial beach elevation.
lineMorphologyData.stdForeshore	1D Array. Standard deviation of sub-aerial beach elevation.
lineRawRectifiedData	<i>MATLAB</i> structure containing raw rectified data.
lineRawRectifiedData.Amat	Raw dataset. The amplitude data for each lidar return.
lineRawRectifiedData.offlineMat	Raw dataset. The distance cross-shore from the downline vector for each lidar return.
lineRawRectifiedData.Rmat	Raw dataset. The distance cross-shore the from point P1 along the vector defined by points P1 and P2 for each lidar return.
lineRawRectifiedData.rotMatrix	The rotation matrix used to rectify the point cloud in space.
lineRawRectifiedData.Tmat	Raw dataset. The time each lidar return was collected.
lineRawRectifiedData.Xmat	Raw dataset. The x-value for each lidar return in SOCS.
lineRawRectifiedData.Ymat	Raw dataset. The y-value for each lidar return in SOCS.
lineRawRectifiedData.Zmat	Raw dataset. The z-value for each lidar return in SOCS.
lineRawRectifiedNoiseFilter	<i>MATLAB</i> structure containing data indicating if a return in the rectified dataset is noise and should be filtered out.
lineRawRectifiedNoiseFilter.isNoise	A matrix of true/false values for each lidar return, indicating if that return is noise.
lineRunupCalc	<i>MATLAB</i> structure containing runup line variables.
lineRunupCalc.downLineIndex	1D array. The index into each scanline of the gridded dataset at which the runup line falls.
lineRunupCalc.downLinePosition	1D array. The cross-shore distance in meters of the runup line location.
lineRunupCalc.griddedDataIsWater	2D array. Boolean value to indicate whether data stored in lineGriddedData is water.
lineRunupCalc.isQAQC	Boolean to indicate if the stored runup line and data has been QAQC'd in <i>RunupTool</i> and reprocessed.
lineRunupCalc.rawRectifiedDataIsWater	2D array. Boolean value to indicate whether data stored in lineRawRectifiedData is water.
lineRunupCalc.Z	1D array. Reference elevation at every bin down lidar scan line.
lineRunupCalc.zDiffCumulative	2D array. Cumulative difference between elevation at a given location and the elevation of the swash runup line.
lineRunupData	Structure. Data and statistics derived from the swash time-series.
lineRunupData.X	Structure. Cross-shore data and statistics from cross-shore coordinate's wave time-series.
lineRunupData.X.FFT	Results from FFT on cross-shore swash time-series.
lineRunupData.X.FFT.amp	1D array. Energy in each wave frequency band of cross-shore swash time-series.

<code>lineRunupData.X.FFT.freq</code>	1D array. Center of wave frequency bands for FFT on cross-shore swash time-series.
<code>lineRunupData.X.FFT.pkPeriodIg</code>	Peak period infragravity wave height ($f < 0.05$ Hz) of cross-shore swash time-series.
<code>lineRunupData.X.FFT.pkPeriodIn</code>	Peak period of sea/swell wave height ($0.05 < f < 0.03$) of cross-shore swash time-series.
<code>lineRunupData.X.FFT.sig</code>	Significant infragravity wave height ($f < 0.05$ Hz) of cross-shore swash time-series.
<code>lineRunupData.X.FFT.sin</code>	Significant sea/swell wave height ($0.05 < f < 0.03$) of cross-shore swash time-series.
<code>lineRunupData.X.r2per</code>	2% exceedance of cross-shore swash time-series.
<code>lineRunupData.X.ts</code>	Full cross-shore swash time-series
<code>lineRunupData.X.tsindx</code>	Index of distance down lidar scan line for full cross-shore swash time-series.
<code>lineRunupData.Z</code>	Structure. Elevation data and statistics from swash foreshore elevation time-series.
<code>lineRunupData.Z.FFT</code>	Results from FFT on swash foreshore elevation time-series.
<code>lineRunupData.Z.FFT.amp</code>	1D array. Energy in each wave frequency band of swash foreshore elevation time-series.
<code>lineRunupData.Z.FFT.freq</code>	1D array. Center of wave frequency bands for FFT on swash foreshore elevation time-series.
<code>lineRunupData.Z.FFT.pkPeriodIg</code>	Peak period infragravity wave height ($f < 0.05$ Hz) of swash foreshore elevation time-series.
<code>lineRunupData.Z.FFT.pkPeriodIn</code>	Peak period of sea/swell wave height ($0.05 < f < 0.03$) of swash foreshore elevation time-series.
<code>lineRunupData.Z.FFT.sig</code>	Significant infragravity wave height ($f < 0.05$ Hz) of swash foreshore elevation time-series.
<code>lineRunupData.Z.FFT.sin</code>	Significant sea/swell wave height ($0.05 < f < 0.03$) of swash foreshore elevation time-series.
<code>lineRunupData.Z.r2per</code>	2% exceedance of swash foreshore elevation time-series.
<code>lineRunupData.Z.ts</code>	Full swash foreshore elevation time-series.
<code>lineWaveGaugeData</code>	Cell array of structures with data and statistics from water surface elevation time-series at virtual wave gauge locations.
<code>lineWaveGaugeData{gaugeIdx}</code>	Structure. Location specifications, raw data, and statistics from water surface elevation for a given wave gauge.
<code>lineWaveGaugeData{gaugeIdx}.FFT</code>	Structure. Results of FFT on water surface elevation time-series at virtual gauge location.

<code>lineWaveGaugeData{gaugeIdx}.FFT.amp</code>	1D array. Energy in each wave frequency band of water surface elevation time-series.
<code>lineWaveGaugeData{gaugeIdx}.FFT.freq</code>	1D array. Center of wave frequency bands for wave energy density spectra.
<code>lineWaveGaugeData{gaugeIdx}.FFT.pkPeriodIg</code>	Peak period infragravity waves ($f < 0.05$ Hz, $\pi > 20$ s) of water surface elevation time-series.
<code>lineWaveGaugeData{gaugeIdx}.FFT.pkPeriodIn</code>	Peak period sea/swell waves ($0.05 < f < 0.03$, $\pi \in 3-20$ s) of water surface elevation time-series.
<code>lineWaveGaugeData{gaugeIdx}.FFT.sig</code>	Significant infragravity wave height ($f < 0.05$ Hz) of water surface elevation time-series.
<code>lineWaveGaugeData{gaugeIdx}.FFT.sin</code>	Significant sea/swell wave height ($0.05 < f < 0.03$) of water surface elevation time-series.
<code>lineWaveGaugeData{gaugeIdx}.gaugeLocation</code>	Structure. Location specifications, for a given wave gauge.
<code>lineWaveGaugeData{gaugeIdx}.gaugeLocation.downLineDistance</code>	Distance down the lidar scan line to the cross-shore location of the virtual wave gauge from the scanner.
<code>lineWaveGaugeData{gaugeIdx}.gaugeLocation.lat</code>	Latitude (degrees north) for the location of the virtual wave gauge.
<code>lineWaveGaugeData{gaugeIdx}.gaugeLocation.lon</code>	Longitude (degrees east) for the location of the virtual wave gauge.
<code>lineWaveGaugeData{gaugeIdx}.gaugeLocation.offLineDistance</code>	Distance off the lidar scan line to the alongshore location of the virtual wave gauge.
<code>lineWaveGaugeData{gaugeIdx}.gaugeLocation.X</code>	FRF cross-shore coordinate for the location of the virtual wave gauge.
<code>lineWaveGaugeData{gaugeIdx}.gaugeLocation.Y</code>	FRF alongshore coordinate for the location of the virtual wave gauge.
<code>lineWaveGaugeData{gaugeIdx}.hs</code>	Total significant wave height ($H_s = 4 * \sigma$) of water surface elevation time-series.
<code>lineWaveGaugeData{gaugeIdx}.lidarLocation</code>	Structure. Location specifications of water surface elevation time-series collection location.
<code>lineWaveGaugeData{gaugeIdx}.lidarLocation.downLineDistance</code>	Distance down the lidar scan line to the cross-shore location of water surface elevation time-series.
<code>lineWaveGaugeData{gaugeIdx}.lidarLocation.downlineDistanceStd</code>	Variation in the distance down the lidar scan line to the cross-shore location of water surface elevation time-series.
<code>lineWaveGaugeData{gaugeIdx}.lidarLocation.downLineIndex</code>	Index of virtual wave gauge's and water surface elevation time-series's cross-shore location in <code>lineCoreDat</code> arrays.
<code>lineWaveGaugeData{gaugeIdx}.lidarLocation.offLineDistance</code>	Distance off the lidar scan line to the alongshore location of water surface elevation time-series.

<code>lineWaveGaugeData{gaugeIdx}. lidarLocation. offLineDistanceStd</code>	Variation in the distance off the lidar scan line to the alongshore location of water surface elevation time-series.
<code>lineWaveGaugeData{gaugeIdx}. lidarLocation.X</code>	FRF cross-shore coordinate for location of water surface elevation time-series collection.
<code>lineWaveGaugeData{gaugeIdx}. lidarLocation.Xstd</code>	Standard deviation in cross-shore of lidar beam location for water surface elevation time-series collection.
<code>lineWaveGaugeData{gaugeIdx}. lidarLocation.Y</code>	FRF alongshore coordinate for location of water surface elevation time-series collection.
<code>lineWaveGaugeData{gaugeIdx}. lidarLocation.Ystd</code>	Standard deviation in alongshore of lidar beam location for water surface elevation time-series collection.
<code>lineWaveGaugeData{gaugeIdx}. time</code>	Start time for the water surface elevation time-series.
<code>lineWaveGaugeData{gaugeIdx}. ts</code>	Structure. Full water surface elevation time-series.
<code>lineWaveGaugeData{gaugeIdx}. ts.time</code>	1D array. Sample times of the water surface elevation time-series.
<code>lineWaveGaugeData{gaugeIdx}. ts.z</code>	1D array. Time-series of water surface elevations.
<code>lineWaveGaugeData{gaugeIdx}. wlmax</code>	Maximum of the water surface elevation time-series.
<code>lineWaveGaugeData{gaugeIdx}. wlmean</code>	Mean of the water surface elevation time-series.
<code>lineWaveGaugeData{gaugeIdx}. wlmin</code>	Minimum of the water surface elevation time-series.
<code>loadFramescansFromRXP</code>	allows for reading consistent data from multiple raw data files in a single function call.
<code>monthlyDir</code>	The path to a directory containing monthly netCDF data files, described in Section 5.
<code>monthlyDir</code>	The path to the directory that will receive the netCDF files of quality controlled data products.
<code>name</code>	The names of the .rxp files used to generate the bare earth DEM
<code>numFrameScans</code>	The number of individual frame scans that are acquired in each complete dataset (three for the FRF).
<code>numLineScans</code>	The number of individual linescans that are acquired in each complete dataset (one for the FRF).
<code>offlineDist</code>	For each grid cell, the mean offline distance of every point that falls in the cell.
<code>offlineDistStd</code>	For each grid cell, the standard deviation of the offline distance of every point that falls in the cell.
<code>offlineMat</code>	The “offline” distance for each point.
<code>paramFile</code>	Path to the parameters file.
<code>paramFile</code>	Path to the parameters file.
<code>paramsFile</code>	The path to the parameters file.

paramsFile	The path to the parameters file.
paramsFile	The path to the parameters file.
paramsFolder	The path to the directory that contains parameters files.
pointvalues	A cell array of values to retrieve for each lidar return.
pointvalues	A cell array of values to retrieve for each lidar return.
QAQCmeta	A data structure containing metadata about QA/QC.
rawRectifiedDataIsWater	An $m \times n$ masking matrix indicating whether each point in the rawRectifiedData matrices are water (i.e., 1 means the point is water; 0 means it is not). In this matrix, n is equal to the maximum number of returns in any scan line (i.e., it will have the same dimensions as the rawRectifiedData matrices).
reflectanceData	The reflectance data from the hourly scan, interpolated onto the same regular grid.
reprocessingQueue	The path to the text file that contains the list of datasets that have been quality controlled and are ready for reprocessing.
returns	A string indicating which lidar returns to read from all files.
returns	A string indicating which lidar returns to read from all files.
Rmat	The downline distance for each point.
rotationMatrix	The rotation matrix calculated during coregistration.
rotationMatrix	The rotation matrix calculated during coregistration.
rotFolder	Path to the directory containing rotation matrices (as .mat files).
rotFolder	The path to the directory that contains rotation matrices generated from baseline scans.
rotFolder	Path to the directory containing rotation matrices (as .mat files).
rotMatrix	The rotation matrix calculated during coregistration.
runupEdited	1D array, created by <i>RunupTool</i> . Stores the location of the new swash runup line. Sorted as the index in downLineDistance arrays in lineCoreDat. Along with runupIndices makes a sparse representation of <i>RunupTool</i> edits.
runupFile	The path to the file containing the new runup line (as created by <i>RunupTool</i>). This will be the *.line.data.mat file.
runupIndices	1D array, created by <i>RunupTool</i> . Stores the indices of the edits along the full runup line. Along with runupEdited makes a sparse representation of <i>RunupTool</i> edits.
runupVariable	The variable in the file that contains the new runup line.

savecoreg	Boolean flag indicating whether or not to save coregistration information in a *.coreg.mat file.
savecoreg	Boolean flag indicating whether or not to save coregistration information in a *.coreg.mat file.
tGPSVector	A 1xm vector containing the average GPS time for each scan line.
tGPSVectorRange	A structure containing the min and max GPS time for each scan number.
Tmat	GPS time at which return pulse was collected.
type	A string describing the type of QA/QC that has been performed on the data. Currently, only “runup” is supported.
varargin	A variable number of arguments, based on the value of type. For a type of runup, the following must be included:
varargin	Optional parameters that are used for the reprocessing of QA/QC'd datasets.
waterLine	The x-coordinate of the alongshore location of the water line for each lidar phi angle, determined using the waterfilter.m function.
waterLineStd	The standard deviation of the location of the water line.
webFolder	An optional directory that will receive copies of the images generated by the current processing dataset, useful for updating a website that provides status updates of the system.
Xmat	x-coordinates in rectified space.
xs	The cross-shore X vector defining the DEM grid, which spans from x = 39.975 m x = 179.975 m with 0.1 m spacing.
Ymat	y-coordinates in rectified space.
Z	An mx1 array of the z-value at the runup position for each scan line.
zDiffCumulative	Cumulative count of the number of times the data at a cross-shore location was classified as water above various running-minimum filters.
zGrid	An mxn matrix of z-values. Cells that contain no points are filled with NaN.
zGridInterp	An m×n z-value with small gaps of NaN cells filled with linearly interpolated values. A maximum of 10 consecutive NaN cells will be filled with interpolated values.
Zmat	z-coordinates in rectified space.

REPORT DOCUMENTATION PAGE				Form Approved OMB No. 0704-0188	
The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing the burden, to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.					
1. REPORT DATE November 2018		2. REPORT TYPE Final Report		3. DATES COVERED (From - To)	
4. TITLE AND SUBTITLE Continuously Operating Dune-Mounted Lidar System at the Field Research Facility: A Report Detailing Lidar Collection, Processing, Evaluation, and Product Development				5a. CONTRACT NUMBER	
				5b. GRANT NUMBER	
				5c. PROGRAM ELEMENT NUMBER	
6. AUTHOR(S) Katherine Brodie, Tristan Dyer, Nicholas Spore, Richard Slocum, Annika O'Dea, Tucker Whitesides, and Alexander Renaud				5d. PROJECT NUMBER 468430	
				5e. TASK NUMBER	
				5f. WORK UNIT NUMBER	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) (see reverse)				8. PERFORMING ORGANIZATION REPORT NUMBER ERDC/CHL SR-18-3	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Coastal Field Data Collection (CFDC) Program U.S. Army Engineer Research and Development Center Coastal and Hydraulics Laboratory Vicksburg, MS 39180-6199				10. SPONSOR/MONITOR'S ACRONYM(S) CHL	
				11. SPONSOR/MONITOR'S REPORT NUMBER(S)	
12. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.					
13. SUPPLEMENTARY NOTES					
14. ABSTRACT The Coastal and Hydraulics Laboratory, Field Research Facility (FRF), stationary lidar tower in Duck, NC, provides hourly, high-density point cloud data of the ocean, foreshore, and dune. This report documents the system and approaches used to collect, process, and analyze the lidar data. Data collection consists of a single 30-minute linescan and three framescans collected every hour. The linescans generate a two-dimensional cross-shore timeseries of the dry beach, swash, inner-surf zone waves, and water levels. The three framescans are used to generate a three-dimensional point-cloud and digital elevation model of the dry beach, quantifying changes along 500 meters of beach. Data are processed in near-real-time using automated algorithms that translate raw data into accurate geo-referenced coordinates by co-registering all data to fixed objects. Linescan data provide quantitative measurements of beach morphology change on very short timescales (seconds to minutes), as well as time-series statistics that include wave run-up elevations, inner-surf zone waves, and setup. This system improves upon the FRF capacity to observe the beach and surfzone onshore of the sandbar in near-real-time. These measurements will therefore serve as a crucial component of the FRF Coastal Model Testbed, providing detailed morphological and hydrodynamic observations at the shoreline.					
15. SUBJECT TERMS Coastal engineering, Data collection platforms, Electronic data processing, Optical data processing, Optical radar					
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Katherine Brodie
Unclassified	Unclassified	Unclassified	SAR	152	19b. TELEPHONE NUMBER (Include area code) 252-261-6840

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

Coastal and Hydraulics Laboratory
Field Research Facility
U.S. Army Engineer Research and Development Center
1261 Duck Road
Duck, NC 27949

Department of Civil, Construction, and Environmental
Engineering
North Carolina State University
2501 Stinson Drive
Raleigh, NC 27695

School of Civil and Construction Engineering
101 Kearney Hall
Oregon State University
Corvallis, OR 97331