FINAL REPORT

Simulation, Signal Extraction, and Augmented Visualization for 3D BOSS data

APL-UW buried-object scanning sonar (BOSS) data processing framework overview

SERDP Project MR18-1051

MARCH 2020

Timothy Marston
The Applied Physics Laboratory,
University of Washington

SERDP
DOD ▪ EPA ▪ DOE

Page Intentionally Left Blank

Page Intentionally Left Blank

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
03/10/2020	SERDP Framework Overview	7/19/2018 - 7/19/2021

4. TITLE AND SUBTITLE
Simulation, Signal Extraction, and Augmented Visualization for 3D BOSS data:

APL-UW buried-object scanning sonar (BOSS) data processing framework overview

5a. CONTRACT NUMBER
18-C-0066

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Timothy Marston

5d. PROJECT NUMBER
MR18-1051

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
The Applied Physics Laboratory, University of Washington
1013 NE 40th st.
Seattle, WA 98105

8. PERFORMING ORGANIZATION REPORT NUMBER
MR18-1051

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Strategic Environmental Research and Development Program
4800 Mark Center Drive, Suite 17D03
Alexandria, VA 22350-3605

10. SPONSOR/MONITOR'S ACRONYM(S)
SERDP

11. SPONSOR/MONITOR'S REPORT NUMBER(S)
MR18-1051

12. DISTRIBUTION/AVAILABILITY STATEMENT
DISTRIBUTION STATEMENT A. Approved for public release: distribution unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
The goal of this code is to allow users to:
ÅParse binary data files captured by a buried object scanning sonar into a simple Matlab structure containing raw sonar data, pulse and system information.
ÅPerform basic operations like pulse compression and real - aperture beamforming to improve the SNR and detect regions of interest in the data
ÅGeo - locate detected regions in latitude and longitude, so that different looks of the some target captured by different scans can be compared
ÅPerform phase - preserving SAS processing on regions of interest so that the acoustic color spectra can be observed, allowing feature extraction for machine learning and target classification.

15. SUBJECT TERMS
buried-object scanning sonar (BOSS), data processing, framework

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Timothy Marston
UNCLASS	UNCLASS	UNCLASS	UNCLASS	42	19b. TELEPHONE NUMBER *(Include area code)* 206-265-1777

Page Intentionally Left Blank

Background

The data processing “framework” developed at APL for processing volumetric, downward looking sonar data has evolved from exposure to data captured by multiple types of systems:

- Three unique designs of the buried object scanning sonar (“BOSS”) designed by edgetech with different array geometries and platforms:
  - A boat mounted linear array with no INS and distant transducer
  - Two parallel linear arrays mounted on a very unstable AUV with a high quality INS
  - A stable towfish with low-end INS but many aiding sensors, and alternating transducer configuration (The Multi-Sensor Towbody: MuST).
- A custom system designed by NSWC-PCD

Background

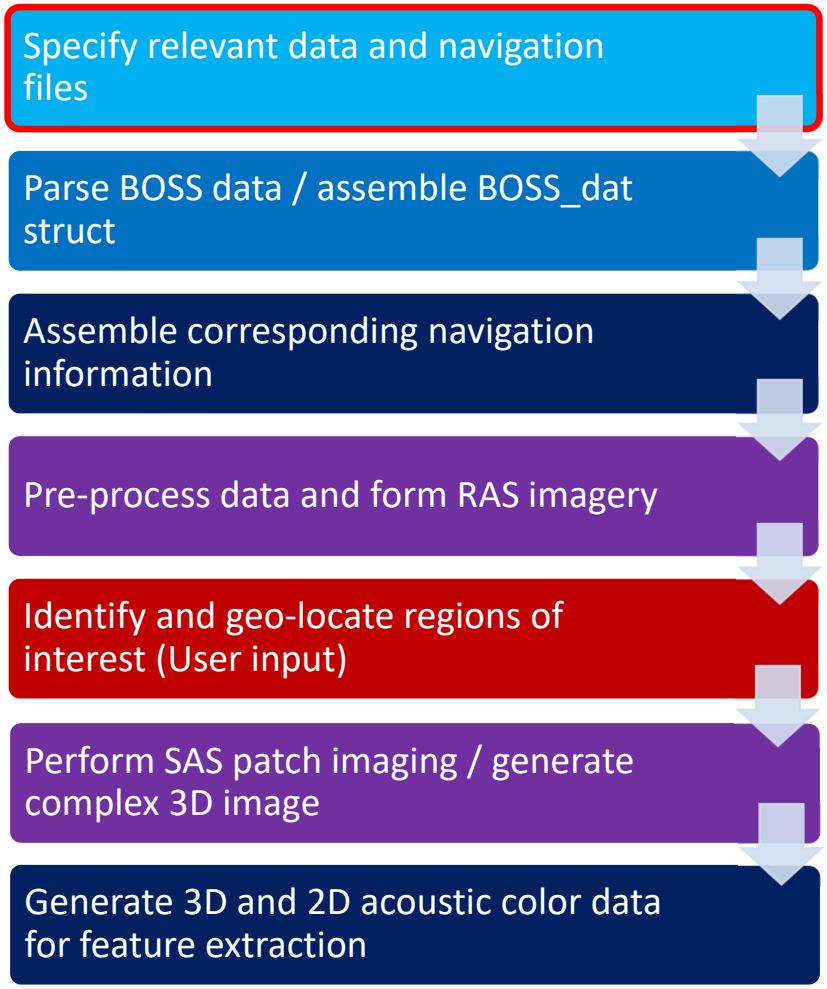
The following discussion of the framework is based around a data exploration and analysis package that I have now delivered to researchers at various institutions for performing beamforming and acoustic color analysis. It is the most polished BOSS processing packaged I have developed, however it does not encapsulate all of the signal processing and capabilities described in the interim.

The MuST is the primary source of data and as the platform evolves so does the signal processing associated with it, so many of the threads of development in this project (i.e. batch processing for SAS) are currently under development. The package described in this powerpoint was selected because it helps describe the general method by which we approach analyzing the data in this project, and is the most distributed, currently, among our associated researchers.

BOSS framework: overview

- The goal of this code is to allow users to:
  - Parse binary data files captured by a buried object scanning sonar into a simple Matlab structure containing raw sonar data, pulse and system information.
  - Perform basic operations like pulse compression and real-aperture beamforming to improve the SNR and detect region's of interest in the data
  - Geo-locate detected regions in latitude and longitude, so that different looks of the some target captured by different scans can be compared
  - Perform phase-preserving SAS processing on regions of interest so that the acoustic color spectra can be observed, allowing feature extraction for machine learning and target classification.
- No special hardware is required, however the image processing toolbox is necessary to access certain features of the toolbox (i.e. 3D visualization).

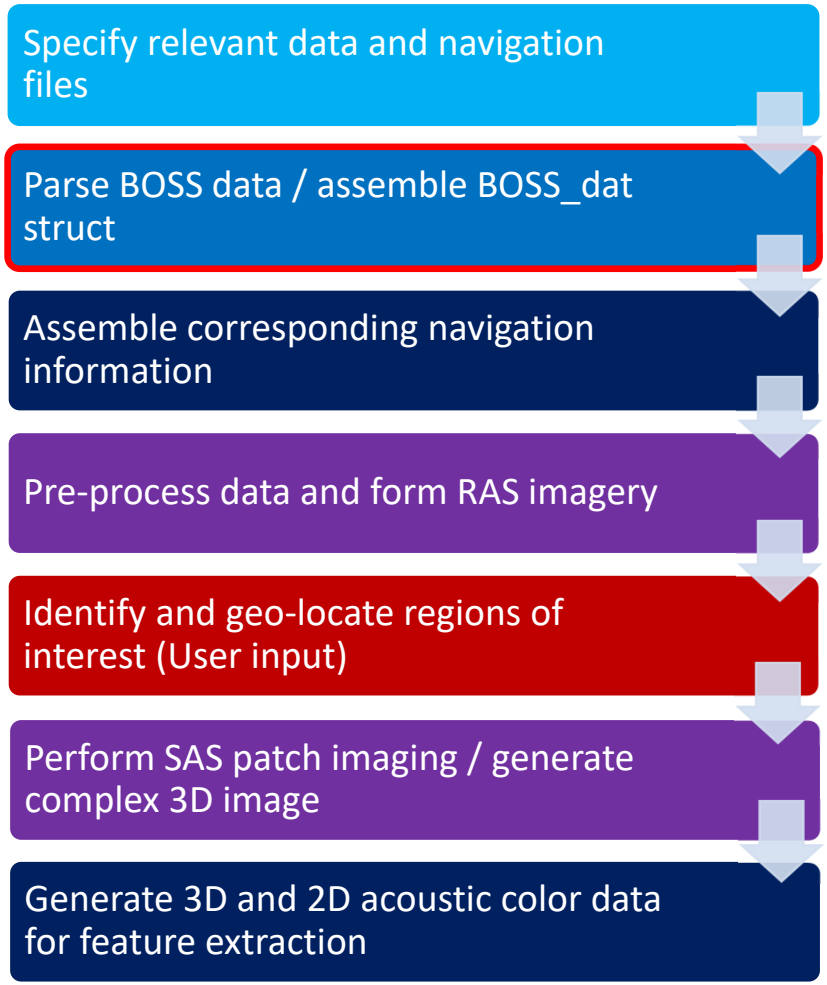
Framework flowchart



Specify relevant data and navigation files

- Data captured from a BOSS system designed by EdgeTech is stored in a binary ".JSF" file, which is specified by the user.
- For the MuST system, navigation information is obtained by post-processing raw sensor data from the INS in Matlab, and thus stored in a .mat file.
- The user needs to identify the nav file relevant for the current JSF. This task has been automated in the most recent versions.

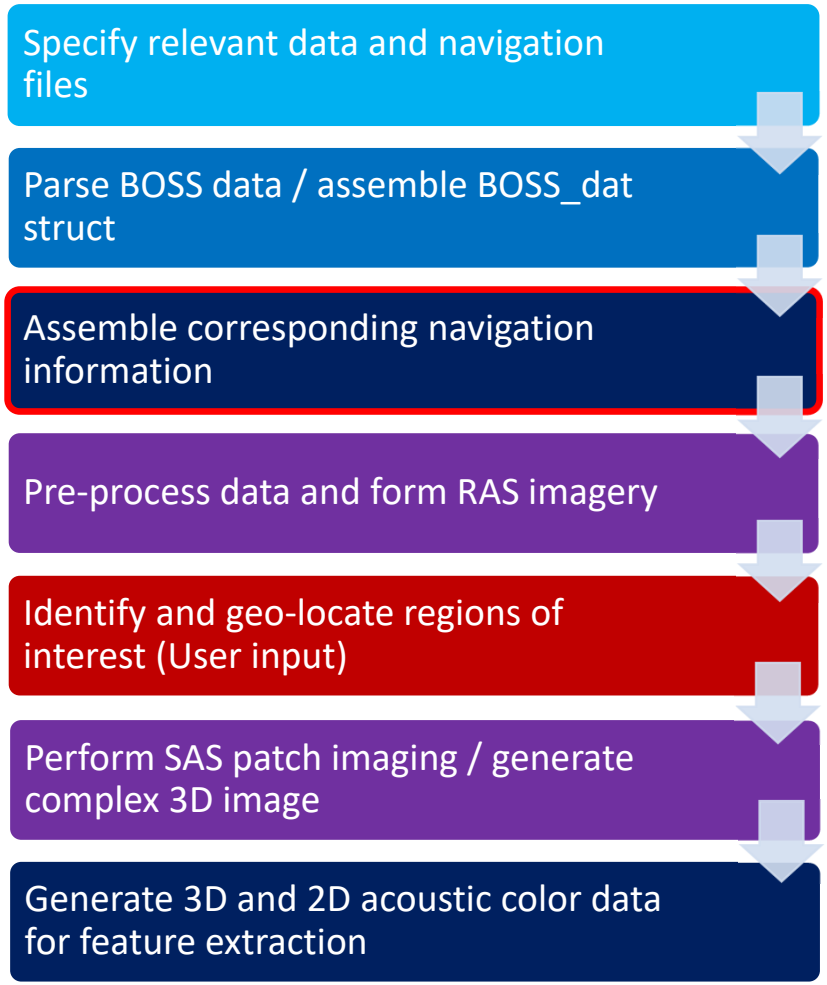
Framework flowchart



Parse BOSS data / assemble BOSS_dat struct

- The binary .JSF file is parsed to form a Matlab structure ("BOSS_dat") containing:
  - Pulse information required to perform pulse compression
  - Blocks of ping data containing raw sonar data, ping timing information in Unix Epoch time (seconds since Jan. 1st, 1970 UTC), source info and various other sonar diagnostic flags
  - Navigation data packages from an INS, if communication between the INS and BOSS system is enabled
  - A message log and total ping count.

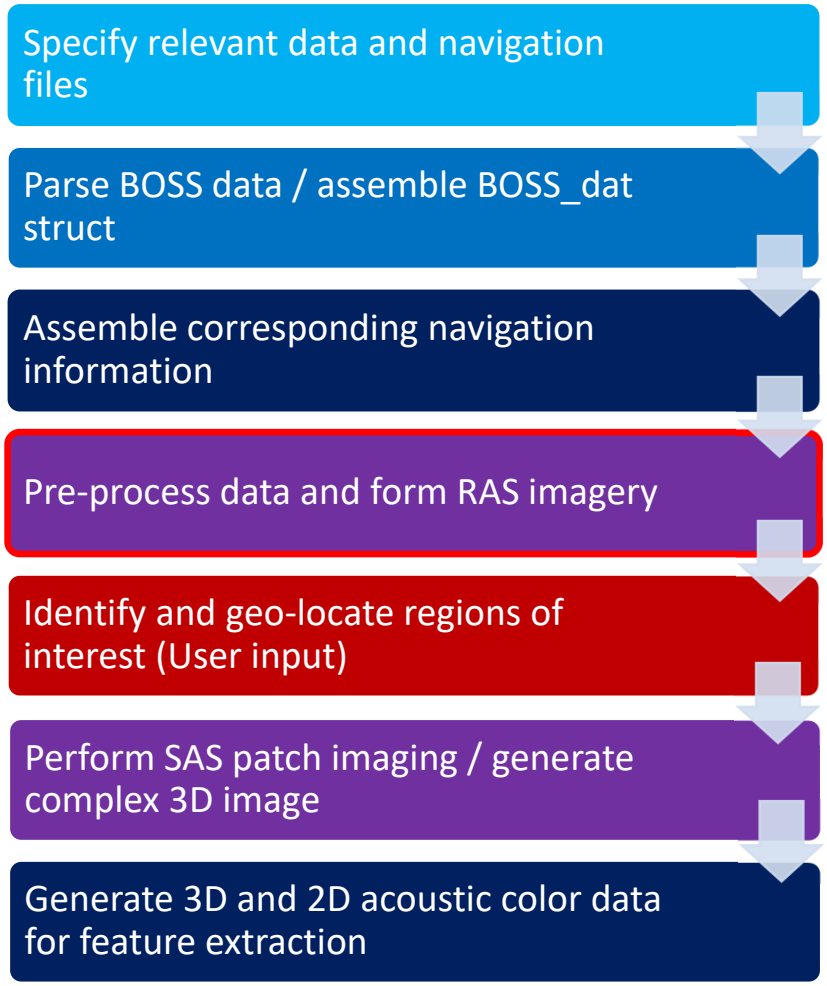
Framework flowchart



Assemble navigation information

- Ping data and navigation data are recorded at different times and rates
- The navigation solution computed in post-processing is searched for overlap with the time history of the JSF file
- Relevant navigation data (e.g. orientation, geo-location in UTM, velocities, depth, etc.) are interpolated onto the ping timings in the JSF file
- A super-structure containing both the BOSS data, interpolated navigation data, and hooks for future dataproducts (e.g. RAS images) is assembled: “MuSTdat”

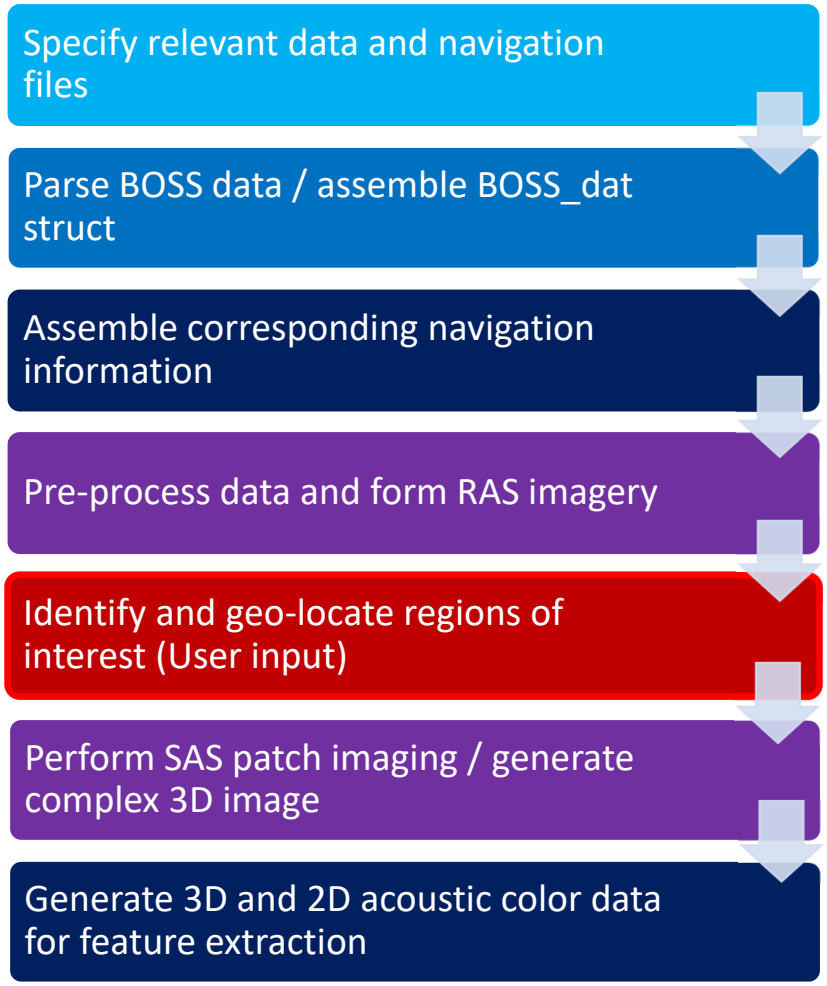
Framework flowchart



Pre-process data and form RAS imagery

- Pulse compression information in MuSTdat.BOSS_dat.pulseInfo is used to pulse-compress the data.
- Data is filtered, base-banded, and decimated to reduce the number of computations required in subsequent operations.
- A structure is generated that contains all of the interpolation and propagation phase tables for real-aperture imaging, and stored in MuSTdat.
- A loop is run through all pings to generate real-aperture (single-ping) images in the across-track dimension. These images are stored in a single 3D array.

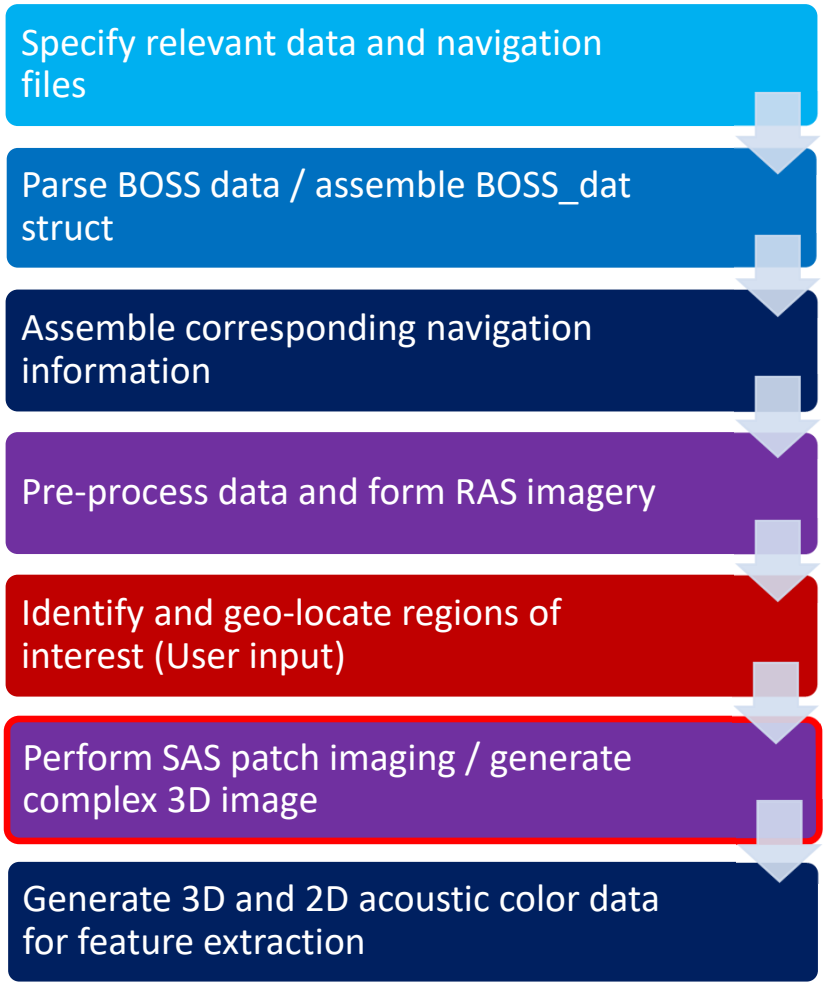
Framework flowchart



Identify and geo-locate regions of interest

- A set of interactive visualizations are supplied to the user to allow the user to manually clock on regions that contain high levels of acoustic energy visible in the RAS images.
- A 3D visualization of the region of interest is supplied to the user for verification of the region-of-interest, which the user can accept or reject.
- From the user's input the actual coordinates in UTM northings, eastings, and depth are generated.

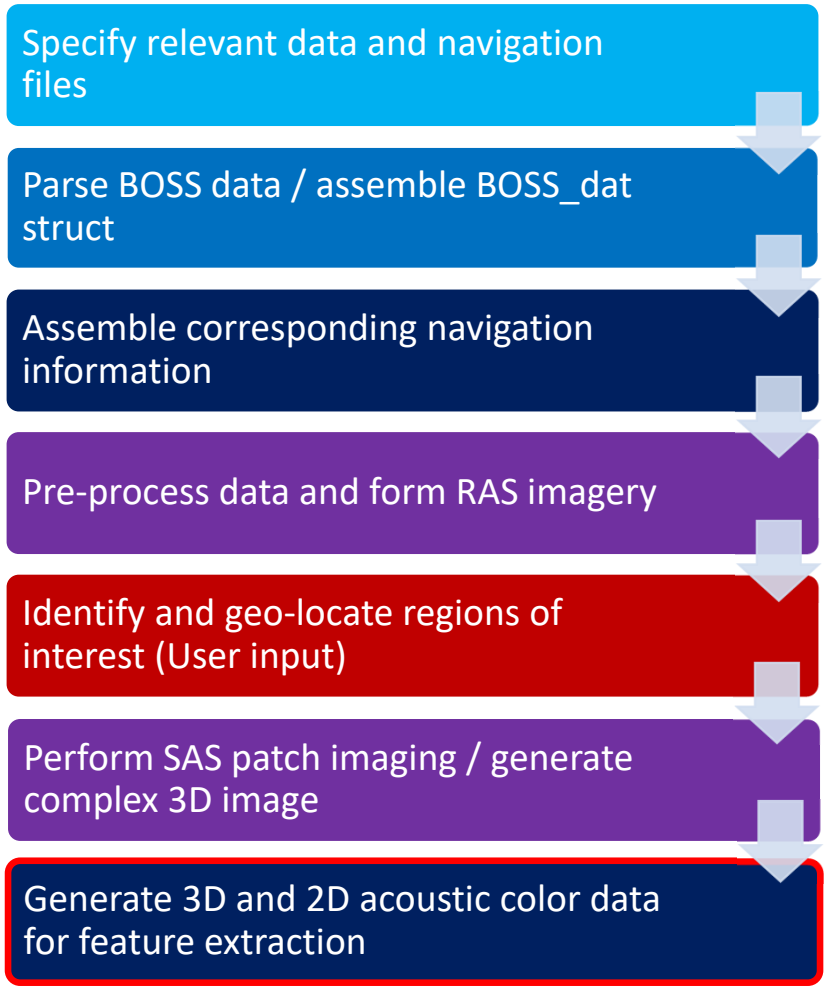
Framework flowchart



Perform SAS patch imaging / generate complex 3D image

- A 3D region in UTM is defined using the user-defined UTM [N, E, depth] coordinate vector, box size and sampling specifications.
- An accelerated back-projection algorithm which is efficient but preserves the quality of spatial-spectral information is applied to generate 3D volumetric imagery in complex double precision

Framework flowchart



Generate acoustic color data

- A 3D FFT is used to generate the spatial spectrum (3D acoustic-color data) corresponding to the 3D ROI.
  - Spatial masking methods will be developed in the future to improve the SNR of this data product.
- A de-noised optimal SNR 2D spectrum through the primary energy in the spectrum is also produced as a sanity check by extracting target information from the raw data and processing it using holographic and spatial filtering techniques.

Framework walkthrough with application

- Sequim bay dataset
- Muddy target field
- Multiple targets visible
- Target imagery and acoustic-color data is desired
- Compare the effects of maximum integration angle

Framework walkthrough with application

Specify relevant data and navigation files

Relevant code snippet:

```
57      % Initialize workspace
58 -    clear all;
59 -    close all;
60
61      % Specifie JSF file and folder:
62 -    JSF = 'Sequim_testDay2_081.jsf';
63 -    JSF_folder = 'C:\Users\tmarston\Documents\MATLAB\Sequim_examples\';
64
65 -    jsfFile = fullfile(JSF_folder, JSF);
66
67      % Load renavigation MuSTnav structure corresponding to the specified JSF
68      % file:
69 -    load EJSF9_10_19_Set_20_to_36renavB.mat;
70
```

User specifies file, folder, and navigation file here

Framework walkthrough with application

Parse BOSS data / assemble BOSS_dat struct

Relevant code snippet:

```
67    % Load renavigation MuSTnav structure corresponding to th
68    % file:
69 —  load EJSF9_10_19_Set_20_to_36renavB.mat;
70
71    % Load BOSS data, and grab the right range of NAV data samples:
72 —  [BOSS_dat] = Read_BOSSJSF(jsfFile);
73 —  [BNAV] = Time_Align_NAV(BOSS_dat, MuSTNAV);
74
75    % RAS beamform to prep for subsequent 3D patch beamforming
76 —  MuSTdat = DW_LOAD_RAS(BOSS_dat, BNAV);
77
```

Boss data is parsed by the function Read_BOSSJSF()

Framework walkthrough with application

Parse BOSS data / assemble BOSS_dat struct

Data structure:

```
K>> BOSS_dat

BOSS_dat =

  struct with fields:

    pulseInfo: [1×1 struct]
    pingChunk: {1×39 cell}
       msgLog: [1×8536 double]
       nPings: 2459
     INS_type: 'ATLAS'
        ATLAS: [1×1 struct]
```

- Pulse compression and transmit signal information
- Ping data blocks
- Message log
- Number of pings
- INS type and navigation data

Framework walkthrough with application

Parse BOSS data / assemble BOSS_dat struct

BOSS_dat sub-structures

```
K>> BOSS_dat

BOSS_dat =

  struct with fields:

    pulseInfo: [1×1 struct]
    pingChunk: {1×39 cell}
       msgLog: [1×8536 double]
       nPings: 2459
     INS_type: 'ATLAS'
        ATLAS: [1×1 struct]
```

```
K>> BOSS_dat.pulseInfo

ans =

  struct with fields:

           pulseName: "eBossApl_5_25_3MS_SR97KHZ_25PctDCFullPower.spf"
       rcvSampleRate: 9.7656e+04
      xmitSampleRate: 500000
             mfCoefs: 903
           pulseGain: 1
       mfFirstFFTCoef: 178
     mfForwardFFTsize: 4096
     mfInverseFFTsize: 2048
        mfGoodSamples: 3676
            xmitSize: 1500
           xmitDelay: 250
         maxPingRate: 83.3330
           numPulses: 1
    numMatchedFilters: 0
         numVGARamps: 0
           numPhases: 0
           xmitPulse: [1500×1 int16]
               mfilt: [1×4096 double]
```

Framework walkthrough with application

Parse BOSS data / assemble BOSS_dat struct

BOSS_dat sub-structures

```
K>> BOSS_dat

BOSS_dat =

  struct with fields:

    pulseInfo: [1×1 struct]
    pingChunk: {1×39 cell}
       msgLog: [1×8536 double]
       nPings: 2459
     INS_type: 'ATLAS'
        ATLAS: [1×1 struct]
```

```
K>> BOSS_dat.pingChunk{1}

ans =

  struct with fields:

             ptime: [1×64 double]
              pnum: [1×64 int32]
               mpx: [1×64 double]
              data: [2380×64×64 int16]
        sampleRate: 9.7656e+04
    discardSamples: 0
        markNumber: 0
        dataFormat: 4
        NumChannls: 64
               srn: [4×1 uint8]
```

Framework walkthrough with application

Parse BOSS data / assemble BOSS_dat struct

BOSS_dat sub-structures

```
K>> BOSS_dat

BOSS_dat =

  struct with fields:

    pulseInfo: [1×1 struct]
    pingChunk: {1×39 cell}
       msgLog: [1×8536 double]
       nPings: 2459
     INS_type: 'ATLAS'
        ATLAS: [1×1 struct]
```

```
K>> BOSS_dat.ATLAS

ans =

  struct with fields:

       telMG: [1×3000 double]
       telLN: [1×3000 double]
       telID: [1×3000 double]
        time: [1×3000 double]
    nav_MODE: [1×3000 double]
     nav_VAL: [1×3000 double]
         lat: [1×3000 double]
         lon: [1×3000 double]
       depth: [1×3000 double]
         alt: [1×3000 double]
        roll: [1×3000 double]
       pitch: [1×3000 double]
        head: [1×3000 double]
          vx: [1×3000 double]
          vy: [1×3000 double]
          vz: [1×3000 double]
          wx: [1×3000 double]
          wy: [1×3000 double]
          wz: [1×3000 double]
          ax: [1×3000 double]
          ay: [1×3000 double]
          az: [1×3000 double]
       cur_x: [1×3000 double]
       cur_y: [1×3000 double]
           c: [1×3000 double]
```

Framework walkthrough with application

Assemble corresponding navigation information

Relevant code snippet:

```
70
71      % Load BOSS data, and grab the right range of NAV data sample
72 -    [BOSS_dat] = Read_BOSSJSF(jsfFile);
73 -    [BNAV] = Time_Align_NAV(BOSS_dat, MuSTNAV);
74
75      % RAS beamform to prep for subsequent 3D patch beamforming
76 -    MuSTdat = DW_LOAD_RAS(BOSS_dat, BNAV);
77
78      %% Get ROI for beamforming:
79 -    xyzO = Select_target_locationMD(MuSTdat);
```

Time alignment / interpolation with renav data performed by Time_Align_NAV()

Framework walkthrough with application

Assemble corresponding navigation information

Output navigation structure: BNAV

```
K>> BNAV

BNAV =

  struct with fields:

    Heading: [1×3000 double]
      Pitch: [1×3000 double]
       Roll: [1×3000 double]
     PosXYZ: [3000×3 double]
       Time: [1×3000 double]
          c: [1×3000 double]
        alt: [1×3000 double]
        Cbn: [3×3×3000 double]
        lat: [3000×1 double]
        lon: [3000×1 double]
    UTMzone: '10 U'
```

PosXYZ is in [Northings, Eastings, Depth] format, and the UTM zone is given as the last field of the struct

Framework walkthrough with application

Pre-process data and form RAS imagery

Relevant code snippet:

```
72 —    [BOSS_dat] = Read_BOSSJSF(jsfFile);
73 —    [BNAV] = Time_Align_NAV(BOSS_dat, MuSTNAV);
74
75      % RAS beamform to prep for subsequent 3D patch beamforming
76 —    MuSTdat = DW_LOAD_RAS(BOSS_dat, BNAV);
77
78      %% Get ROI for beamforming:
79 —    xyzO = Select_target_locationMD(MuSTdat);
80
81      %% 3D Beamform
82 —    IMdat = BOSS_patch(MuSTdat, xyzO, 20);
```

Pulse compression is applied, real aperture beamforming tables are generated, and real aperture (RAS) images are produced by DW_LOAD_RAS().

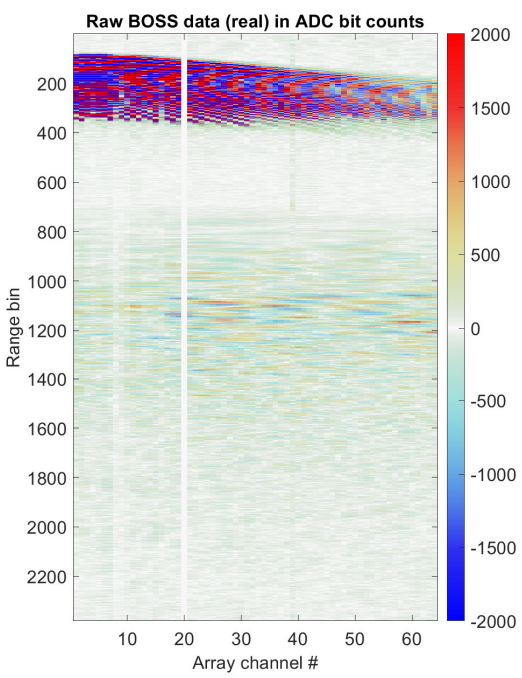
A new structure, MuSTdat, which contains all the BOSS, navigation, RAS, and RAS table data is output.

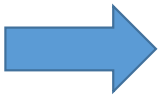
This struct is used as the primary input for the remaining significant operations.

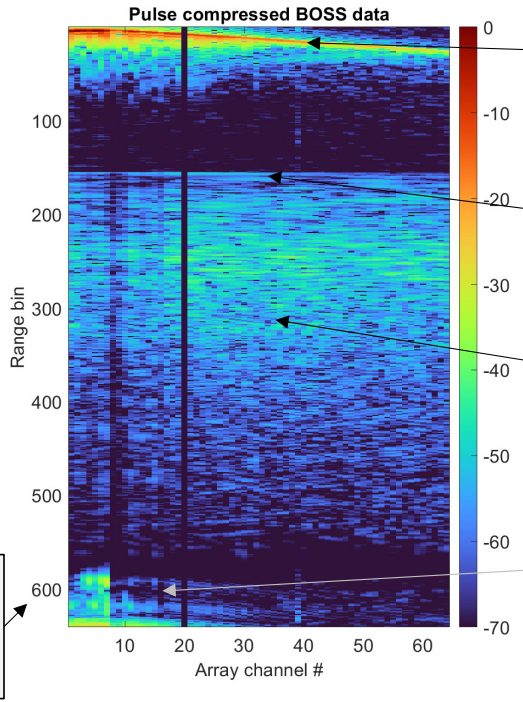
Framework walkthrough with application

Pre-process data and form RAS imagery

Pulse compression & filtering: before & after (single ping, all channels)







Decimated data has fewer range samples

Direct-path signal from transducers to the array

Sediment first bounce

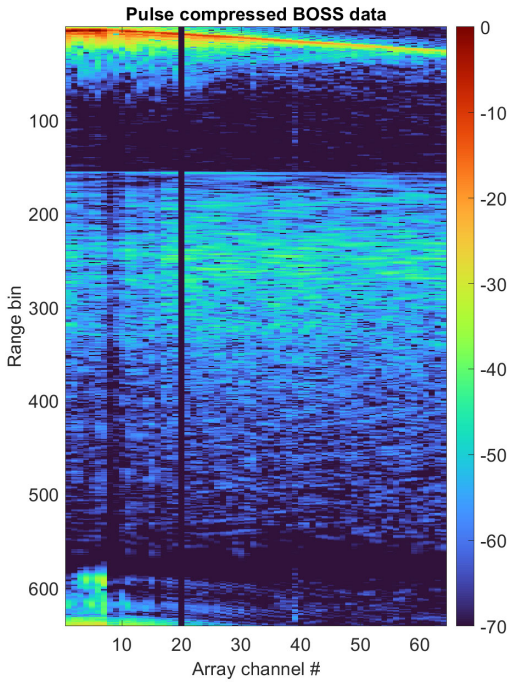
Volume and off-axis scattering

Pulse compression wrap-around artifacts

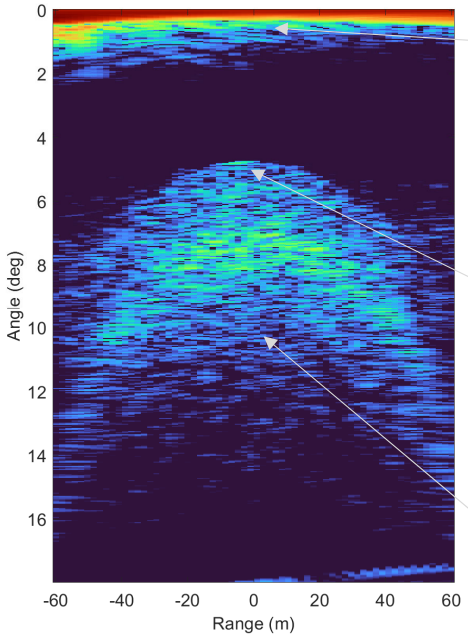
Framework walkthrough with application

Pre-process data and form RAS imagery

RAS image generated from pulse-compressed data



RAS image in polar coordinates



Direct-path signal from transducers to the array

Sediment first bounce. Note that flat sediment interfaces appear like inverted parabolas in polar coordinates.

Volume and off-axis scattering

Framework walkthrough with application

Pre-process data and form RAS imagery

MuSTdat and RAS data structures

```
K>> MuSTdat

MuSTdat =

  struct with fields:

       model: [1×1 mustSystem]
       Array: [1×1 struct]
    BOSS_dat: [1×1 struct]
        BNAV: [1×1 struct]
      mfilts: [1×1 struct]
         VLA: 0
         RAS: [1×1 struct]
```

```
K>> MuSTdat.RAS

ans =

  struct with fields:

       pij: [531×64×2459 double]
       imX: [531×64 double]
       imY: [531×64 double]
       imZ: [531×64 double]
      imTH: [531×64 double]
       imR: [531×64 double]
     rvect: [1×531 double]
     avect: [1×64 double]
     ptime: [1×2459 double]
       mpx: [1×2459 double]
     nfrqs: [640×1 single]
    nsamps: [1×640 double]
        w0: 8.8238e+04
       c_0: 1.4921e+03
```

3D RAS array:
MuSTdat.RAS.pij

Framework walkthrough with application

Identify and geo-locate regions of interest (User input)

Relevant code:

```
74
75      % RAS beamform to prep for subsequent 3D patch beamforming
76 -    MuSTdat = DW_LOAD_RAS(BOSS_dat, BNAV);
77
78      %% Get ROI for beamforming:
79 -    xyzO = Select_target_locationMD(MuSTdat);
80
81      %% 3D Beamform
82 -    IMdat = BOSS_patch(MuSTdat, xyzO, 20);
83      % Display 3D image
84 -    figure
85 -    clf
```

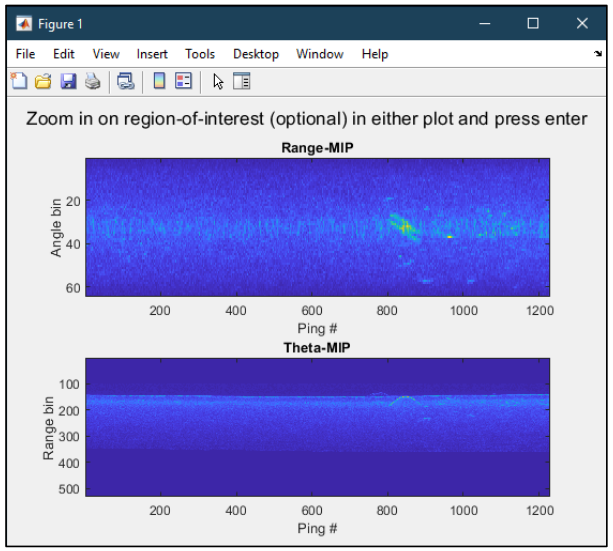
Target selection routine initiated by select_target_locationMD() operating on the MuSTdat struct.

Framework walkthrough with application

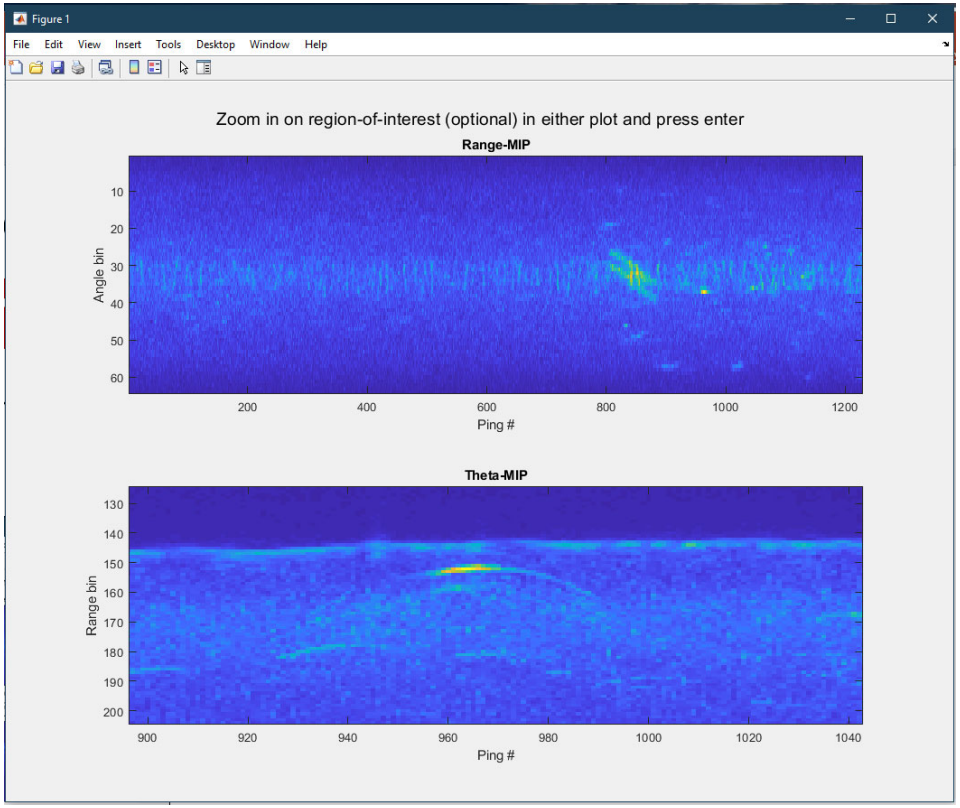
Identify and geo-locate regions of interest (User input)

Example user interaction:

- User is shown a figure with MIP's through range and angle, and prompted to resize the figure, zoom on region of interest in either MIP, and press enter.



User re-sizes the figure window to be larger and zooms in on the theta (profile) MIP to show a prominent target arc

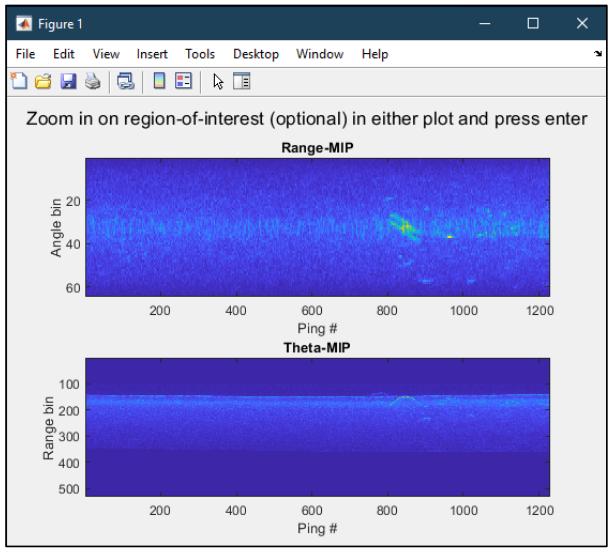


Framework walkthrough with application

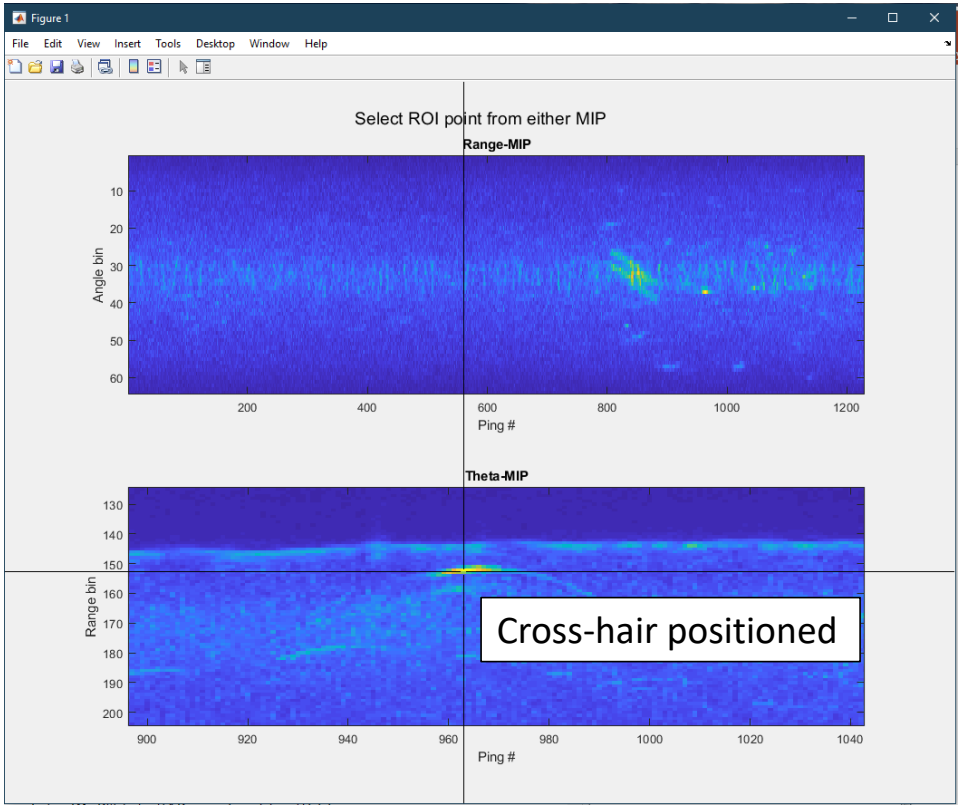
Identify and geo-locate regions of interest (User input)

Example user interaction:

- User is shown a figure with MIP's through range and angle, and prompted to resize the figure, zoom on region of interest in either MIP, and press enter.



User presses "enter" and positions the newly appearing cross-hairs over the prominent target arc



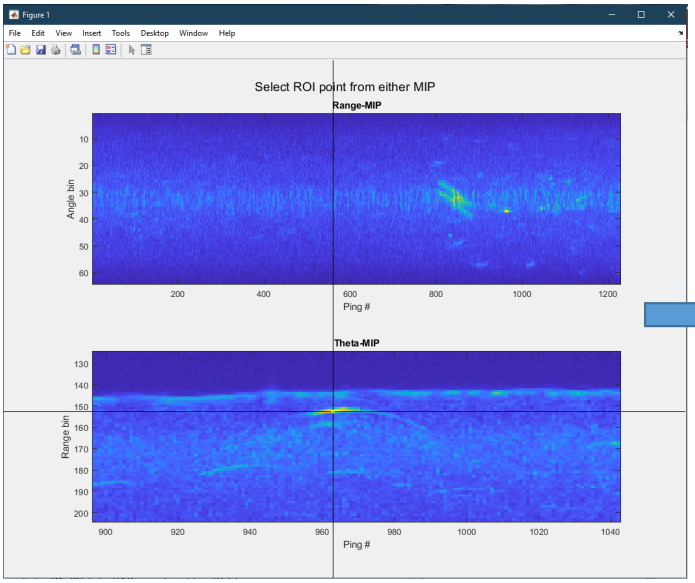
Framework walkthrough with application

Identify and geo-locate regions of interest (User input)

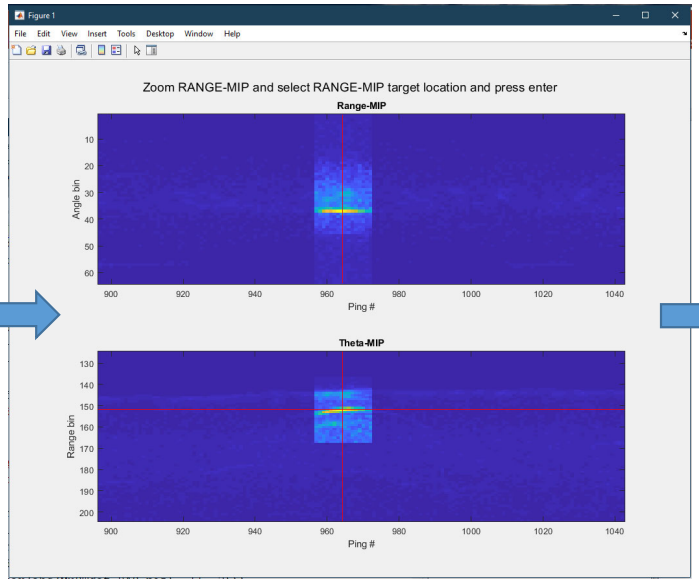
Example user interaction:

- The selected ROI is highlighted in the chosen MIP and the user is invited to define the ROI in the other volume dimension. A spatial mask is applied in the volume to make potential targets more visible in the remaining dimension.

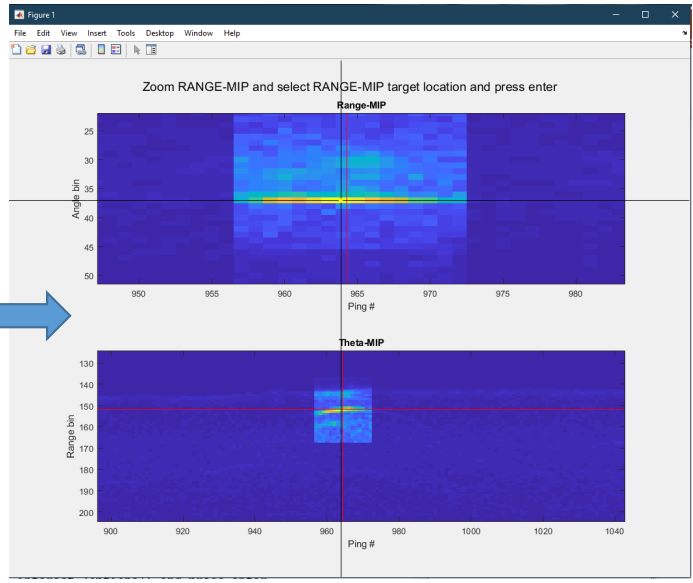
Initial ROI selection by user



Region highlight in remaining dim.



User selection of ROI in remaining dim.



Framework walkthrough with application

Identify and geo-locate regions of interest (User input)

Example user interaction:

User is supplied with an interactive volumetric MIP of the RAS ROI, and prompted to either proceed or restart the ROI selection process.

- If the user is satisfied with the ROI, the coordinates of the target are output as UTM [northings, eastings, depth] in the variable xyzO

ROI interactive volume

Keyboard prompt and output location

```
Selected region. Hit enter to confirm / all other keys = start over
>> xyzO

xyzO =

   1.0e+06 *

    0.4987
    5.3238
   -0.0000
```

Framework walkthrough with application

Perform SAS patch imaging / generate complex 3D image

Relevant code:

```
80
81      %% 3D Beamform
82 -    IMdat = BOSS_patch(MuSTdat, xyzO, 20);
83      % Display 3D image
84 -    figure
85 -    clf
86 -    load ice
87      % h = volshow(fftshift(abs(fftn(pij3))).^0.5,...
```

The ROI is imaged using the function BOSS_patch(), which accepts the MuSTdat struct, the location of the ROI (xyzO), and several other optional parameters (i.e. integration angle, shown here, but also box size and resolution).

Framework walkthrough with application

Perform SAS patch imaging / generate complex 3D image

Relevant code:

```
80
81      %% 3D Beamform
82      iANG = 20;          % Integration angle: +/- 20 deg.
83      rMAX = 10;          % Max range to target (default: max. in time data
84      bxsz = [1, 1, 3];   % Box size in meters. Default: [2, 2, 2]
85      brez = 0.02;        % Box resolution in meters. Default: 0.025 (2.5 cm)
86
87      IMdat = BOSS_patch(MuSTdat, xyzO, iANG, rMAX, bxsz, brez);
88      % Display 3D image
89      figure
90      clf
91      load ice
```

Example with custom values for resolution, box size, and maximum range to target.

Framework walkthrough with application

Perform SAS patch imaging / generate complex 3D image

Results: 3D image data and axes are output in the “IMdat” data structure:

```
>> IMdat

IMdat =

  struct with fields:

     bsX: 2
     bsY: 2
     bsZ: 2
      dR: 0.0250
    bXYZ: [3×531441 double]
    img3: [81×81×81 single]
```

Sizes in [x, y, z] dimensions

Voxel edge length

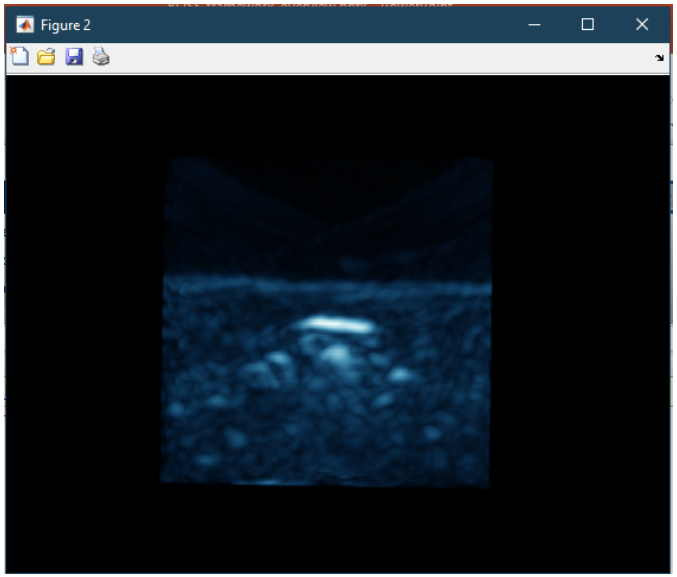
Voxel coordinates in UTM

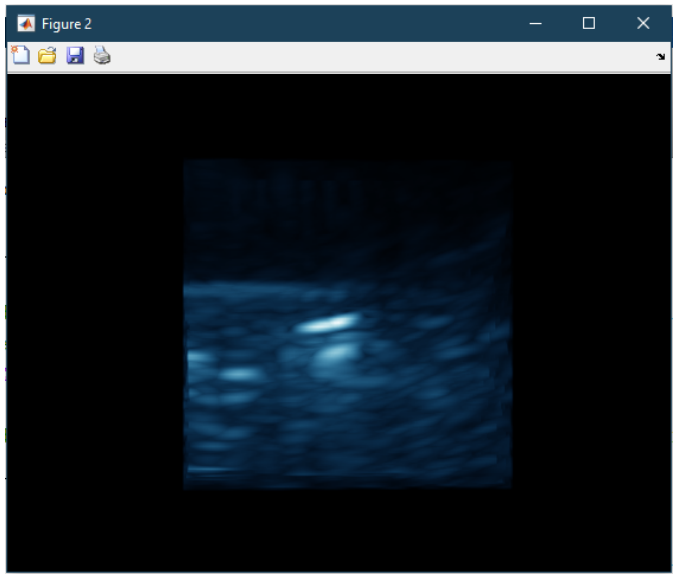
Complex image matrix

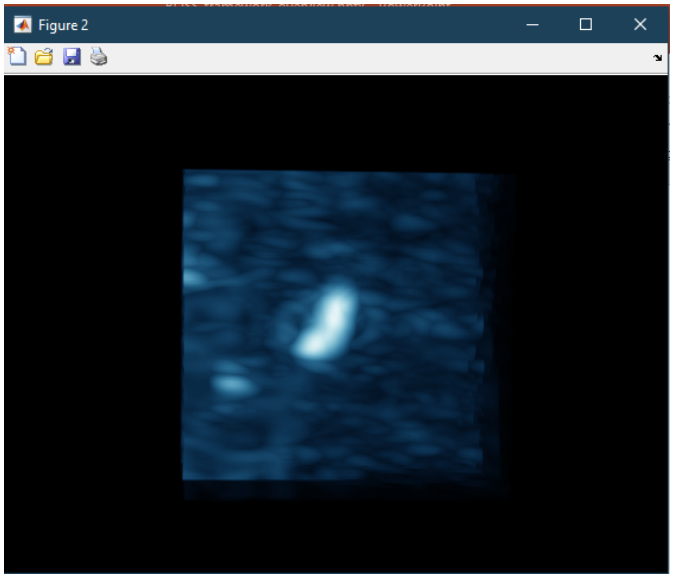
Framework walkthrough with application

Perform SAS patch imaging / generate complex 3D image

Interactive 3D volume MIP rendering, viewed from x, y & z directions:



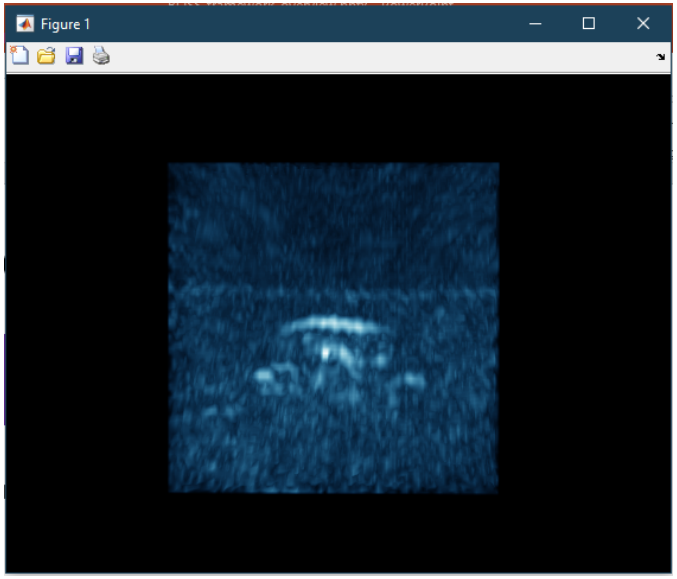


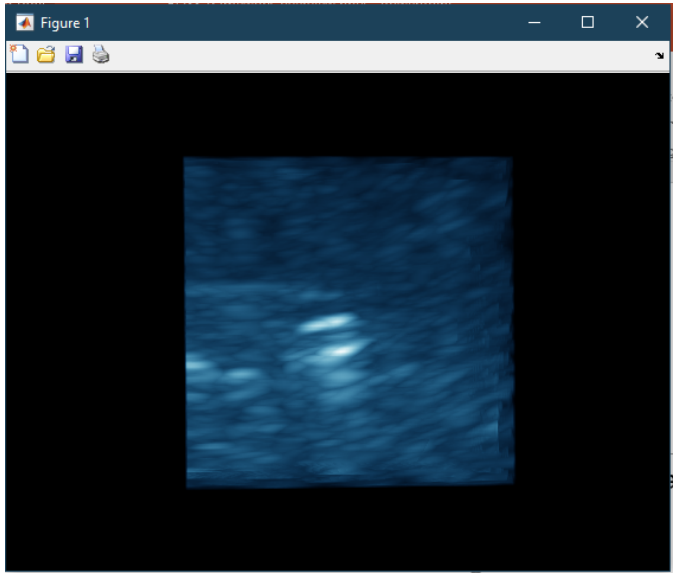


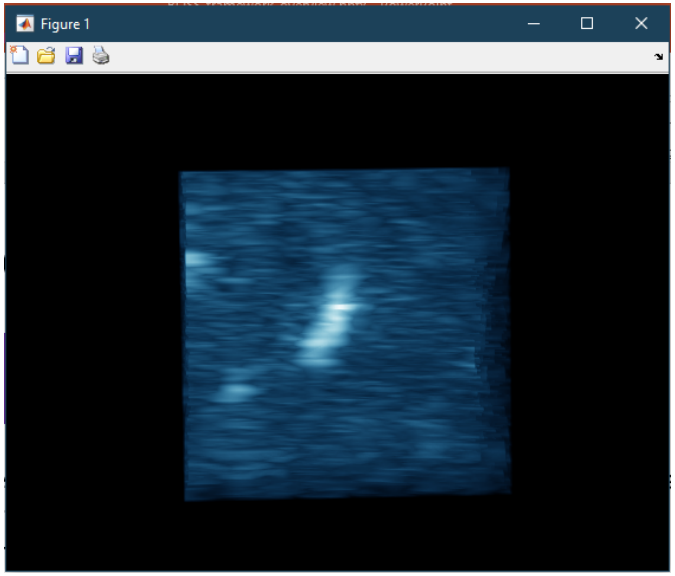
Framework walkthrough with application

Perform SAS patch imaging / generate complex 3D image

Example with extended +/- 45 degrees of angular integration. The image is noisier because the sampling doesn't support this level of angular integration without aliasing, but the resolution is also higher and more features of the target, such as the curved leading edge, are resolved.







Framework walkthrough with application

Perform SAS patch imaging / generate complex 3D image

Relevant code:

```
92
93      %% Dispay 3D acoustic color
94 -    figure
95 -    clf
96 -    load ice
97 -    h = volshow(fftshift(fftshift(abs(fftn(IMdat.img3)), 1), 2).^0.5,...
98          'Renderer', 'MaximumIntensityProjection',...
99          'colormap', ice,...
100         'BackgroundColor', 'k');
101
102     % Alternative: best-plane image & AC
103     % - This approach is what I use to make fast 2D acoustic-color estimates of
104     % regions of interest.
105     % - The only constraint is that you need a relatively decent starting point
106     % for the location of the object.
107 -   figure;
108 -   total_integration_angle = 60;
109 -   ACdat = AC_profile_extract(MuSTdat, xyz0, total_integration_angle/2);
```

3D AC is recovered by computing the 3D FFT of the complex image, IMdat.img3.

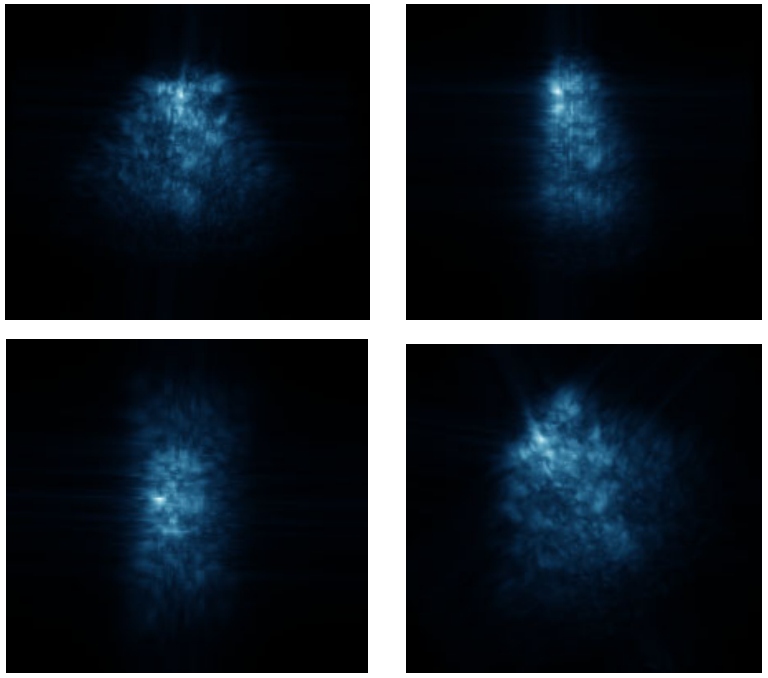
A “sanity check” is performed by extracting and de-noising the raw data corresponding to the maximum SNR slice through the target and computing the AC using that denoised data.

Framework walkthrough with application

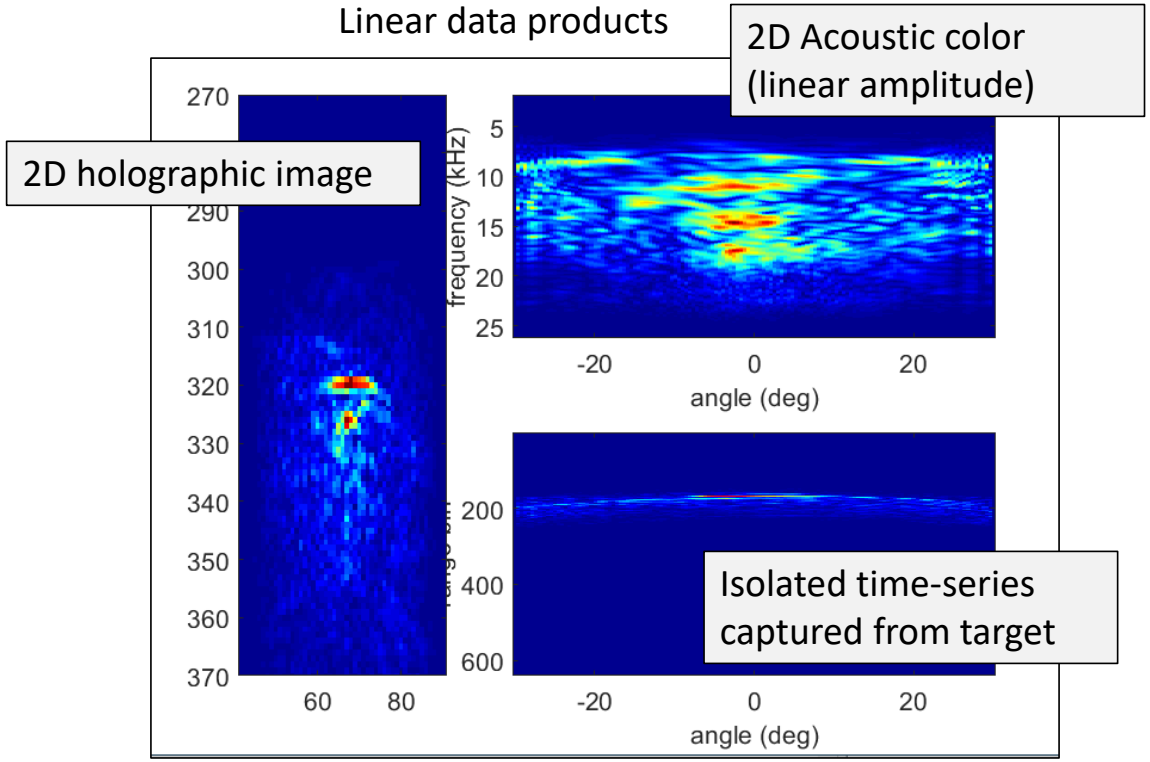
Perform SAS patch imaging / generate complex 3D image

Example 2D and 3D acoustic dataproducts for the beamformed image cube and selected ROI:

3D acoustic color views from different perspectives



Linear data products



Summary and discussion

- A BOSS framework for data exploration, ROI detection, patch beamforming and interaction with complex 3D SAS data (including acoustic color processing) has been developed for researchers.
- The framework can operate on standard computers without NVIDIA GPU's (e.g. apple computers), and provides a simple interface for researchers to use to interact with beamformed and raw sonar data captured by the BOSS system mounted on the MuST.
- Many other significant capabilities have been developed for processing BOSS data (e.g. FFBP for batch SAS imaging, live streaming, etc.), and work is actively underway to integrate these capabilities into the current framework.
- For more information, please contact: Marston@apl.Washington.edu