Sonar Workbench User's Guide

Thomas J. Deal

Naval Undersea Warfare Center, Newport, RI

April 9, 2020

Abstract

Abstract goes here.

Contents

List of Tables

List of Figures

1 Introduction

Sonar Workbench is a suite of Matlab tools for the design and analysis of sonar systems. Much of the content is adapted from *An Introduction to Sonar Systems Engineering* [1], which is recommended as a companion resource for the user interested in understanding more of the theory.

2 Coordinate System and Reference Frames

Sonar Workbench uses a right-handed, Cartesian coordinate system known as North-East-Down (NED), as shown in Fig. 1. In this coordinate system, the first coordinate, x, points north, the second coordinate, y, points east, and the third coordinate, z, points down. Roll, γ, is rotation about the x axis, pitch, θ, is rotation about the y axis, and yaw, ψ, is rotation about the z axis. The NED coordinate system is ideal for underwater applications, because depth is measured downward from the surface, yaw is measured clockwise from north, and pitch is measured relative to the horizontal plane.

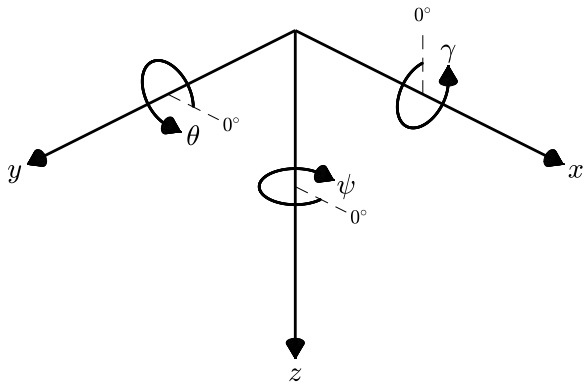

Figure 1: NED coordinate system

Sonar Workbench uses three reference frames: the element frame, the array frame, and the body frame. All frames use the NED coordinate system, and each frame can be located relative to another by a combination of translations[1] and rotations.

The element frame is always located at the center of the element, with the element's maximum response axis aligned with the $+x$ axis. Exceptions to this alignment are the omnidirectional element, which has no maximum response axis, and the linear element, which the user specifies as initially parallel to one of the three axes (x,y,z) in the element frame. For planar piston elements, the element face lies in the element frame y-z plane.

Each element in an array can have arbitrary translation and rotation in the array frame. The array frame origin and orientation is entirely up to the user, but it is typical for planar arrays to be located in center of the array

[1] displacement along the x, y, or z axes

frame's y-z plane and for volumetric arrays' geometric center to be located at the origin of the array frame.

The entire array can also be arbitrarily translated and rotated relative to the body frame. For example, a planar array on the nose of a torpedo might have a simple translation along the body frame x axis, while a flank array might have translations along the body x and y axes plus a rotation ψ about the body frame z axis. Figure 2 shows an example of element, array, and body frames for a conformal flank array.

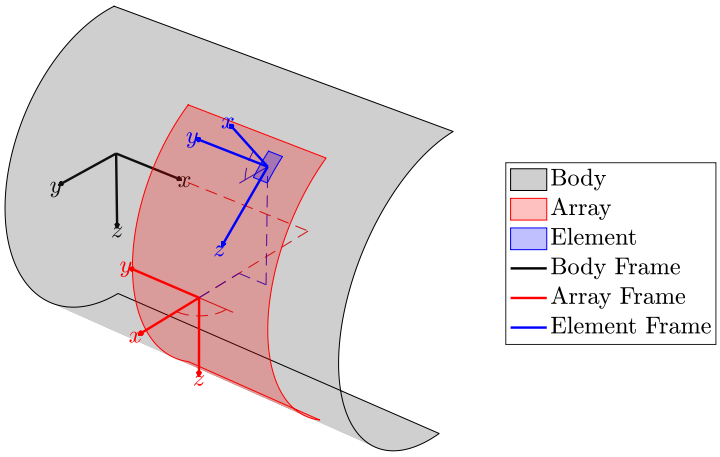

Figure 2: Body, array, and element frames for flank array example

Beam patterns are always computed in the body frame as a function of body azimuth angles ψ and elevation angles θ. Azimuth and elevation angles are measured from the body frame $+x$ axis. For simple analyses, the array and body frames can be aligned and colocated to produce beam patterns in the array frame. In this case, beam pattern angles ψ and θ are measured from the array frame $+x$ axis. More details about element, array, and body frame alignments will be explained in Sections 3 and 4.

3 Elements

The transducer element is the fundamental building block for arrays and the starting point for analysis in Sonar Workbench. For the purpose of generating and analyzing beam patterns, specific electromechanical transduction methods do not need to be modeled; instead, the element's acoustic properties can be captured by modeling vibrations of the element's wetted surface. Hereafter, references to an element's geometry refer to the geometry of the element's wetted surface or face. Sonar Workbench treats each element as a uniformly vibrating surface, which is to say that it only models each surface's fundamental mode of vibration. Analyzing element response requires first, defining the element geometry, and second, evaluating that geometry at a specific acoustic wavelength to produce an element pattern.

3.1 Element definition

Sonar Workbench includes support for the element types listed in Table 1.

Table 1: Included element types

Element Type	**Type String**
Omnidirectional	`'OmnidirectionalElement'`
Uniform Line	`'LinearElement'`
Cosine	`'CosineElement'`
Circular Piston	`'CircularPistonElement'`
Rectangular Piston	`'RectangulerPistonElement'`
Annular Piston	`'AnnularPistonElement'`
Hexagonal Piston	`'HexagonalPistonElement'`

An element structure holds the parameters that define the element geometry. The element structure must contain the `.type` field with a string corresponding to the name of a `.m` file that generates the corresponding element pattern. Table 1 lists the built-in element type strings, but the user can define additional element types by generating their own element pattern generator script with the same interface.

The field `.baffle` dictates whether the element should be baffled by the element frame y-z plane (e.g. arrays mounted to platforms) or unbaffled (e.g. towed arrays, sonobuoys). The omnidirectional, uniform line, and cosine elements can all be used with and without baffling. The piston elements' element patterns are all derived from equations assuming the pis-

ton is mounted in an infinite rigid baffle; therefore, it is recommended that the user set these element's `.baffle=1` for best results. Diffraction effects caused by finite baffle dimensions are beyond the scope of Sonar Workbench.

For visualization purposes, fields `.shapex`, `.shapey`, and `shapez` contain vectors of element shape coordinates. The script `AddElementShape.m` generates these vectors for the element types listed in Table 1. The other fields in the element structure depend on the element type, as listed in Table 2.

Table 2: Element structure fields

Element Type	**Field**	**Description**
Uniform Line	`.L`	length (m)
	`.axis`	aligned axis 'x', 'y', 'z'
Circular Piston	`.a`	radius (m)
Rectangular Piston	`.w`	width (m)
	`.h`	height (m)
Annular Piston	`.a`	outer radius (m)
	`.b`	inner radius (m)
Hexagonal Piston	`.a`	inscribed circle radius (m)

Listing 1 shows the contents of `SampleElement.m`, which defines a rectangular piston element.

Listing 1: `SampleElement.m`

```
%% Element Design
Element.type = 'RectangularPistonElement';
Element.w = lambda/2;                % Element face width, m
Element.h = lambda/4;                % Element face height, m
Element.baffle = 1;                  % Hard Baffle
Element = AddElementShape(Element);
```

The user is free to add additional fields to the element structure for their own purposes. These will be ignored by Sonar Workbench.

3.2 Element patterns

Element geometry, coupled with an acoustic wavelength, defines the element pattern. The element pattern is the element's far-field directional response as a function of wavelength, azimuth and elevation, and it can be thought of as a spatial filter. Elements are assumed to be transducers, capable of transmitting and receiving sound, so there is no distinction between transmit and receive element patterns.

Sonar Workbench uses the acoustic wavelength, λ, to calculate element patterns because it combines the frequency and sound speed into a single term. It is related to sound speed c, frequency f in Hz or ω in rad/s, and wavenumber k in m^{-1} by

$$\lambda = \frac{c}{f} = \frac{2\pi c}{\omega} = \frac{2\pi}{k}.$$

At its most general, the element pattern is the three-dimensional Fourier transform of the element's complex aperture function, $A(\lambda, x, y, z)$,

$$E(\lambda, \theta, \psi) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} A(\lambda, x, y, z) e^{j2\pi\left(\frac{\cos\theta\cos\psi}{\lambda}x + \frac{\cos\theta\sin\psi}{\lambda}y + \frac{\sin\theta}{\lambda}z\right)} dx dy dz, \tag{1}$$

For the piston elements, the element pattern reduces to a two-dimensional Fourier transform, since the element face lies in the y-z plane,

$$E_{piston}(\lambda, \theta, \psi) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} A(\lambda, y, z) e^{j2\pi\left(\frac{\cos\theta\sin\psi}{\lambda}y + \frac{\sin\theta}{\lambda}z\right)} dy dz, \tag{2}$$

and for the uniform line array, it further reduces to a one-dimensional Fourier transform,

$$E_{line}(\lambda, \theta, \psi) = \int_{-\infty}^{\infty} A(\lambda, y) e^{j2\pi\frac{\cos\theta\sin\psi}{\lambda}y} dy, \tag{3}$$

for a line array aligned with the y axis. The finite element extents make the integration limits finite. For the simple elements included with Sonar Workbench, the assumption of uniform surface motion means that the aperture function is real-valued and equal to 1 over the entire element surface. This simplifies the integration for certain element geometries. These integrals have analytic solutions, which Sonar Workbench uses instead of evaluating the integrals numerically.

Element patterns are normalized such that they have unity gain along their maximum response axis. Figure 3 shows element patterns for each of the included element types listed in Table 1 for a wavelength equal to half of the element's maximum dimension. The uniform line element is aligned with the y axis, and the cosine element is aligned with the x axis. Note that the omnidirectional and cosine element patterns do not depend on wavelength.

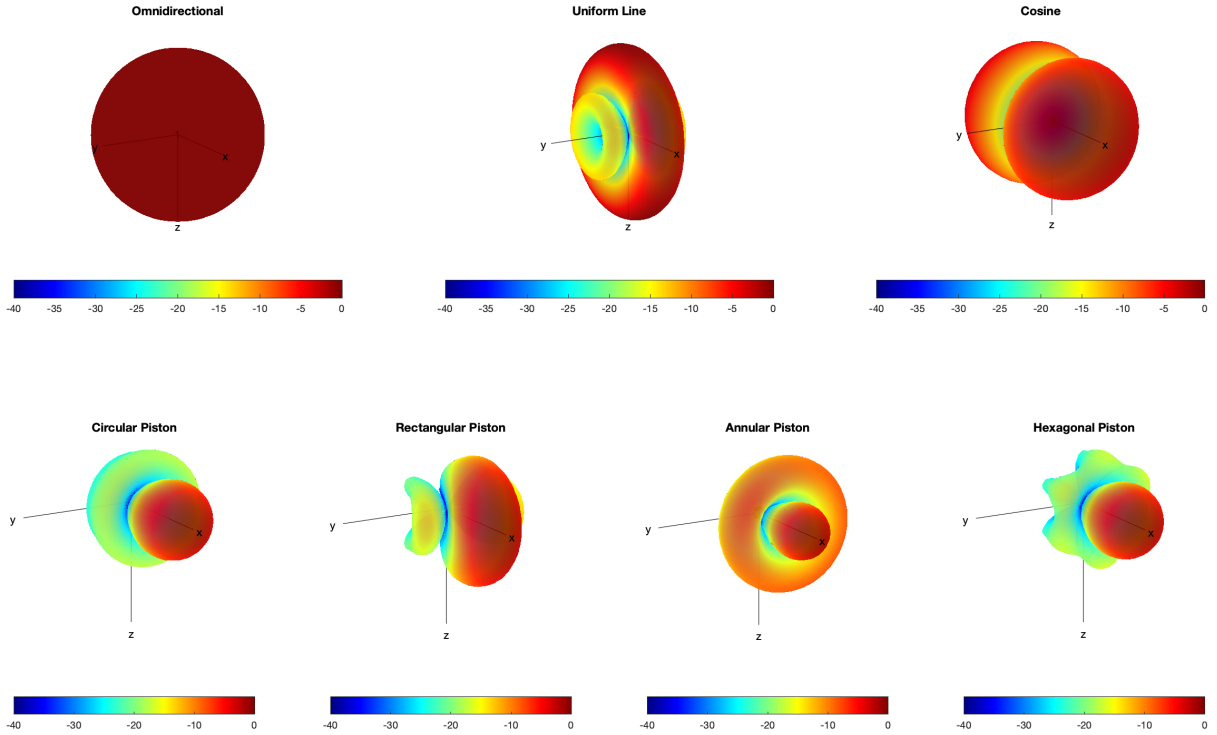

Figure 3: Element patterns for included element types

4 Arrays

An array is a collection of elements. Each element in an array has a position and orientation in the array frame. The array itself also has a position and orientation in the body frame. Sonar Workbench supports arrays with mixed element types, e.g. vector sensor arrays with a mix of hydrophones (omnidirectional elements) and accelerometers (cosine elements). The array structure has fields which contain this information.

The field `.Ne` holds the integer number of elements in the array. The position, orientation, and element type fields are all column vectors of length `Ne`. The user is free to choose any convenient element order, as long as that order is consistent across all fields. Three vectors determine the element positions in the array frame, `.ex`, `.ey`, and `.ez`, all in units of meters. Three additional vectors determine the element orientations in the array frame, `.egamma`, `.etheta`, and `.epsi`, corresponding to roll, pitch, and yaw, in degrees.

The array structure holds an additional 6 fields with scalars corresponding to the array position and orientation in the body frame. These fields are `.ax`, `.ay`, `.az`, `.agamma`, `.atheta`, and `.apsi`, in units of meters and degrees. Table 3 summarizes the fields in the array structure.

Table 3: Array structure fields

Field	Description
`.Ne`	number of elements
`.ex`	element x position vector (m)
`.ey`	element y position vector (m)
`.ez`	element z position vector (m)
`.egamma`	element roll orientation vector (°)
`.etheta`	element pitch orientation vector (°)
`.epsi`	element yaw orientation vector (°)
`.eindex`	element type index (optional)
`.ax`	array x position (m)
`.ay`	array y position (m)
`.az`	array z position (m)
`.agamma`	array roll orientation (°)
`.atheta`	array pitch orientation (°)
`.apsi`	array yaw orientation (°)

4.1 Arrays with uniform element types

When all the array elements are identical, the array structure does not need the `.eindex` field. An example array structure is found in `SampleArray.m`, shown in Listing 2.

Listing 2: `SampleArray.m`

```
%% Array Design
Nw = 5;                                 % Number of elements wide
Nh = 10;                                % Number of elements high
dy = Element.w;                         % Horizontal spacing, m
dz = Element.h;                         % Vertical spacing, m
Array.Ne = Nw*Nh;                       % Number of elements
Array.ex = zeros(Array.Ne,1);           % Element x positions, m
Array.ey = ...
    repmat((-(Nw-1)/2:(Nw-1)/2)'*dy,Nh,1);
                                        % Element y positions, m
Array.ez = ...
    reshape(repmat((-(Nh-1)/2:(Nh-1)/2)*dz,Nw,1),Array.Ne,1);
                                        % Element z positions, m
Array.egamma = zeros(Array.Ne,1);       % Element rolls, deg
Array.etheta = zeros(Array.Ne,1);       % Element elevations, deg
Array.epsi = zeros(Array.Ne,1);         % Element azimuths, deg
Array.ax = 0;                           % Array x position, m
Array.ay = 0;                           % Array y position, m
Array.az = 0;                           % Array z position, m
```

This planar array contains 50 rectangular elements arranged in a grid 5 elements wide by 10 elements high. The geometry is shown in Fig. 4, with elements numbered according to their order in the array structure.

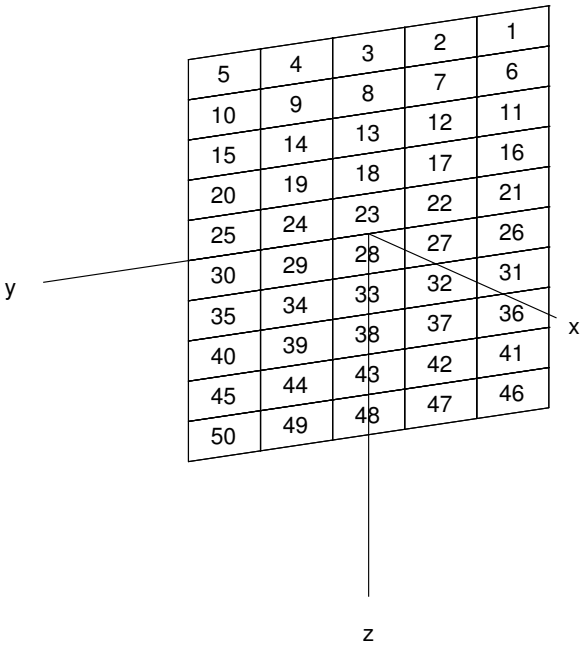

Figure 4: Example rectangular planar array

4.2 Arrays with mixed element types

For mixed-element arrays, the `.eindex` field must be a column vector of length `Ne` integers. The user must first define an element structure array, with one entry for each unique element type. The `.eindex` field for each element must contain the index into the element structure array corresponding to its element type.

An example is included in `SampleVSCardioid.m` for a simple vector sensor. The relevant portion is shown in Listing 3. This vector sensor consists of four elements: a hydrophone and three orthogonal accelerometers. The three accelerometers are identical except for their orientations, so there are a total of two unique elements. The hydrophone is `Element(1)`, and the accelerometer is `Element(2)`. The array structure is named `VS`, and the hydrophone is the first element, followed by the x, y, and z accelerometers. This makes the first entry in `.eindex` equal to 1 and the rest of the entries equal to 2. Fields `.psi` and `.etheta` rotate elements 3 and 4 from the x axis to the y and z axes, respectively.

Listing 3: `SampleVSCardioid.m`

```
%% Define Elements
Element(1).type = 'OmnidirectionalElement';
Element(1).baffle = 0;
Element(1).shapex = [0 0];
Element(1).shapey = [0 0];
Element(1).shapez = [0 0];
Element(2).type = 'CosineElement';
Element(2).baffle = 0;
Element(2).shapex = [1 -1];
Element(2).shapey = [0 0];
Element(2).shapez = [0 0];
%% Define Vector Sensor
VS.Ne = 4;
VS.ex = zeros(VS.Ne,1);
VS.ey = zeros(VS.Ne,1);
VS.ez = zeros(VS.Ne,1);
VS.egamma = zeros(VS.Ne,1);
VS.etheta = [0; 0; 0; 90];
VS.epsi = [0; 0; 90; 0];
VS.ax = 0;
VS.ay = 0;
VS.az = 0;
VS.eindex = [1; 2; 2; 2];
```

5 Beams

Once an array and element(s) have been defined, the user can generate any number of beam patterns by defining a set of wavelength-dependent, complex weights, $w(\lambda)$. Multiple beams can be calculated for the same array by changing these weights. The beam structure must contain these weights in a single field called `.ew`, which is a column vector of length `Array.Ne`.

In conventional (plane wave) beamforming, amplitude weights are used to adjust the width of the beam's main lobe and the magnitude of its sidelobes, and phase weights are used to steer the beam in azimuth and elevation. For a source in the array's acoustic near field, phase weights can also be used to focus the beam at a specified range. In adaptive beamforming, a complex set of weights is chosen to simultaneously maximize one criterion (e.g. signal level) and minimize one or more other criteria (e.g. noise level). Although the details of near-field focusing and adaptive beamforming are beyond the scope of this user's guide, Sonar Workbench can generate beam patterns for any complex weights the user derives.

5.1 Conventional beamforming

Conventional beamforming independently calculates amplitudes, a_i, and phases, ϕ_i to create complex weights $w_i = a_i e^{j\phi_i}$. Listing 4 shows an example of one beam that can be formed for the planar array example from Section 4. This example contains both amplitude and phase weights.

Listing 4: `SampleBeam.m`

```
%% Load Array Definition
SampleArray
%% Amplitude Weights
ea = repmat(chebwin(Nw,30),Nh,1).*reshape(repmat(chebwin(Nh,30)
    ',Nw,1),Array.Ne,1);
% Phase Weights
if ~exist('psi0','var')
    psi0 = 0;
end
if ~exist('theta0','var')
    theta0 = 0;
end
ep = exp(-1i*2*pi*cosd(-theta0).*sind(psi0)/lambda*Array.ey).*
    exp(-1i*2*pi*sind(-theta0)/lambda*Array.ez);
%% Complex Weights
Beam.ew = ea.*ep;
clear ea ep
```

5.1.1 Amplitude weights

Amplitude shading exploits the Fourier transform relationship between the beam's aperture function and its far field beam pattern. Consequently, sonar designers often use weights, or shading coefficients, taken from standard window functions. Examples include Uniform, Hanning, Hamming, Chebyshev, and Raised Cosine. This example uses two Chebyshev windows, one of length 5 in the horizontal direction, and one of length 10 in the vertical direction. The total weight for each element is the product of the two windows evaluated at its coordinate the horizontal and vertical direction.

Since the final beam pattern will be normalized for unity gain along its maximum response axis, the user is not required to normalize the amplitude weights or limit their amplitude. However, it is best to avoid negative amplitude weights and instead represent negative numbers with phase weights.

5.1.2 Phase weights

Without phase weights, i.e. the beam pattern weights are purely real, the beam pattern is unsteered. The user can steer the beam to an arbitrary elevation and azimuth, (θ_0,ψ_0) using the following formula for the i^{th} element's phase,

$$\phi_i(\lambda, \theta_0, \psi_0) = -2\pi \left(\frac{\cos\theta_0 \cos\psi_0}{\lambda} x_i + \frac{\cos\theta_0 \sin\psi_0}{\lambda} y_i + \frac{\sin\theta_0}{\lambda} z_i \right). \quad (4)$$

5.2 Computing beam patterns

When a suitable set of complex weights has been defined, the beam pattern is then calculated as

$$B(\lambda, \theta, \psi) = \sum_{i=1}^{N_e} w_i(\lambda, \theta_0, \psi_0) E_i(\lambda, \theta, \psi) e^{j2\pi\left(\frac{\cos\theta\cos\psi}{\lambda} x_i + \frac{\cos\theta\sin\psi}{\lambda} y_i + \frac{\sin\theta}{\lambda} z_i\right)}, \quad (5)$$

where $E_i(\lambda, \theta, \psi)$ is the element pattern for the i^{th} element, rotated according to the element orientation (γ_i,θ_i,ψ_i), and (x_i,y_i,z_i) are the coordinates of the i^{th} element.

The beam pattern for the example array with no beam steering is shown in Fig. 5. The array is redrawn in that figure with element opacity proportional to amplitude weight.

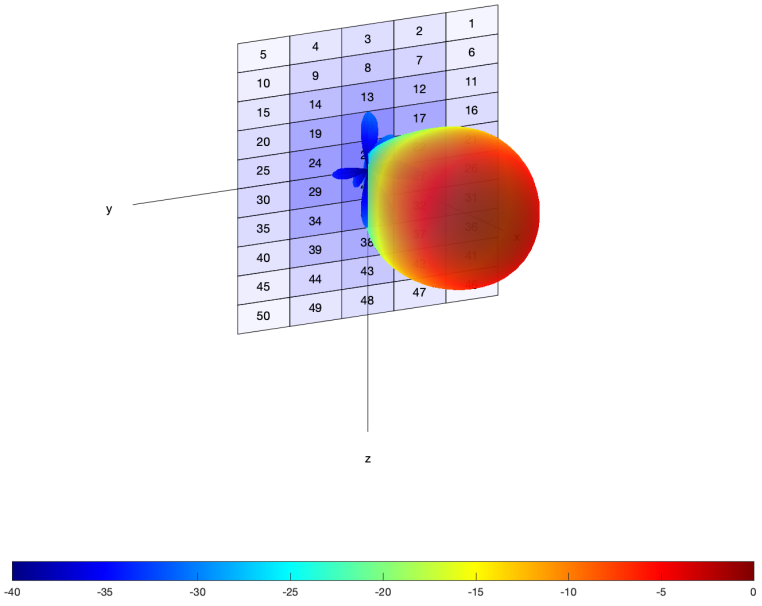

Figure 5: Example beam pattern for rectangular planar array

References

[1] L. J. Ziomek, *An introduction to sonar systems engineering*. Boca Raton, FL: Taylor & Francis/CRC, 1st ed., 2017.