# Sonar Workbench User's Guide

Thomas J. Deal

Naval Undersea Warfare Center, Newport, RI

April 8, 2020

**Abstract**

Abstract goes here.

# Contents

# 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, $\gamma$, is rotation about the $x$ axis, pitch, $\theta$, is rotation about the $y$ axis, and yaw, $\psi$, 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.

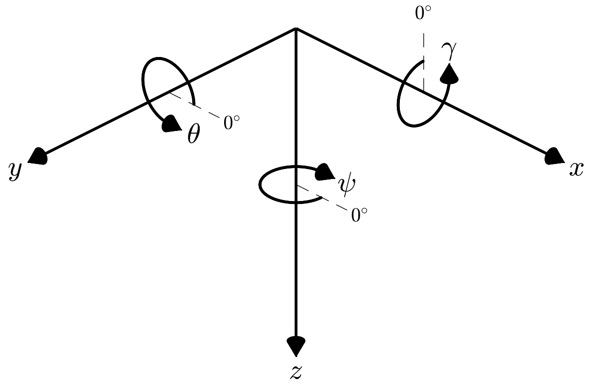

Figure 1: NED coordinate system

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

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

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 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 $\psi$ about the body frame $z$ axis. Figure 2 shows an example of element, array, and body frames for a conformal flank array.

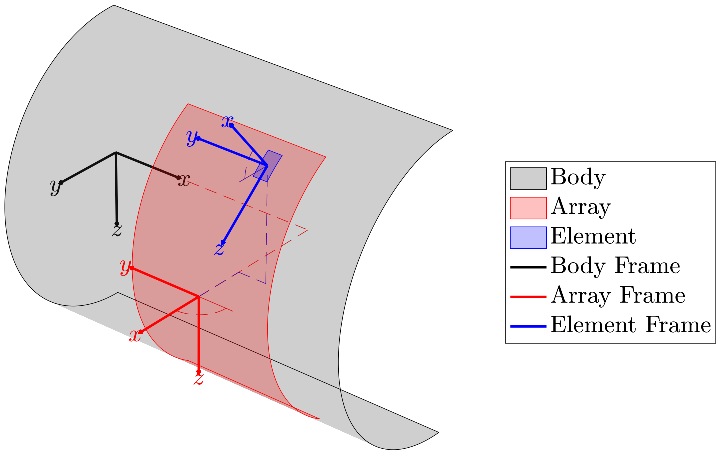

Figure 2: Body, array, and element frames for flank array example

Beam patterns are always computed in the body frame as a function of body azimuth angles $\psi$ and elevation angles $\theta$. 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 $\psi$ and $\theta$ 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

Sonar Workbench includes support for the element types listed in Table 1. It treats each element as a uniformly vibrating surface.

Table 1: Supported 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. The element structure must contain the `.type` field with a string corresponding to a `.m` file that generates the corresponding element pattern. Table 1 lists the built-in element type strings, but the user can easily define additional element types by generating their own element pattern generator script with the same interface. The element `.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). 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) |

# 4 Arrays

# 5 Beams

## References

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