922-20007002

M3 IMB Beamformed Data Format

Document revisions

Version	Date	Written by	Checked by	Approved by
1.0	Aug. 22, 11	AZ	JR/PF	MM
1.1	Oct. 05, 11	AZ	CS	MM
1.2	May. 03, 13	AZ	CS	BC
1.3	Sep. 25, 14	AZ	JR/RH	BC
1.4	Jul. 03, 15	AZ	CS	BC
1.5	Oct 24, 16	JR	AZ	RH

About this document

The information contained in this document is subject to change without prior notice. Kongsberg Mesotech Ltd. shall not be liable for errors contained herein, or for incidental or consequential damages in connection with the furnishing, performance, or use of this document.

Kongsberg Mesotech Ltd.

1598 Kebet Way
Port Coquitlam, BC
V3C 5M5 Canada

Telephone: +1 604 464 8144
Telefax: +1 604 941 5423
www.kongsberg-mesotech.com
Email: km.sales.vancouver@kongsberg.com

KONGSBERG

Table of Contents

Document history

1.0	First revision.
1.1	Modified mounting and beam angle drawing
1.2	Added **bHeadSensorVersion**, **HeadHWStatus**, **fInternalSensorHeading**, **fInternalSensorPitch**, **fInternalSensorRoll**, **sAxesRotatorOffsets**, **nStartElement**, **nEndElement**, **strCustomText1**, **strCustomText2**, **fLocalTimeOffset**
1.3	Added **fHeave** field.
1.4	Added **fPRIMinRange**, **fPRIMaxRange**, **fSoundSpeed_RT.**
1.5	Added **REAL_TIME_SOUND_SPEED**, **PROFILING_DEPTH_TRACKING**, **GPS_QUALITY_PARAS**, **fAltitude**, **fHeightGeoidAbvEllipsoid**, **bTimeSyncMode**, **bSoundSpeedSource**, **bTxPulseType. Removed fSoundSpeed_RT.**

1 DATA FORMAT

1.1 Common Data Type

Data Type	Description
unsigned int16	**16-bit unsigned integer**
unsigned int32	**32-bit unsigned integer**
float	**32-bit floating point**
double	**64-bit floating point**
Ipp32fc	**32-bit floating point complex(I and Q)**
char	**8 bit character**
byte	**8 bit unsigned integer**

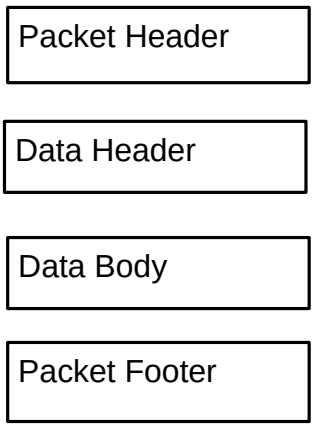
1.2 Constant Structure

Structure	Fields	Data type	Description
MUM_TVG_PARAMS	factorA	unsigned int16	factor A, the spreading coefficient
	factorB	unsigned int16	factor B, the absorption coefficient in dB/km
	factorC	float	factor C, the TVG curve offset in dB
	factorL	float	factor L, the maximum gain limit in dB
M3_OFFSETS	xOffset	float	Translational offset in X axis
	yOffset	float	Translational offset in Y axis
	zOffset	float	Translational offset in Z axis
	xRotOffset	float	Rotational offset about X axis in degrees (Pitch offset)
	yRotOffset	float	Rotational offset about Y axis in degrees (Roll offset)
	zRotOffset	float	Rotational offset about Z axis in degrees (Yaw offset)
	dwMounting	unsigned int32	Mounting orientation. Fixed head mounting parameters from the Deployment Configuration, for information only. The mounting orientation is fully described by

			the xRotOffset, yRotOffset and zRotOffset fields. If dwVersion < 5 0: mounting Down or Aft 1: mounting Up or Fore If dwVersion >=5 0: Custom; 1: Forward; 2: Forward Inverted; 3: Roll Right; 4: Roll Left; 5: Upward; 6: Upward Inverted; 7: Downward; 8: Downward Inverted
M3_ROTATOR_OFFSETS	fOffsetA	float	Rotator offset A in meters
	fOffsetB	float	Rotator offset B in meters
	fOffsetR	float	Rotator offset R in meters
	fAngle	float	Rotator angle in degrees
REAL_TIME_SOUND_SPEED	fRaw	float	Raw sound speed in m/s directly from sensor readings
	fFiltered	float	Filtered sound speed in m/s after a median filter on the raw values
	fApplied	float	Applied sound speed in m/s after the filtering and thresholding, and will be used for sonar data processing
	fThreshold	Float	A user predefined threshold in m/s used to get the applied value
PROFILING_DEPTH_TRACKING	nRangeDeepLimitPercentage	byte	The percentage of the maximum range for auto depth algorithm to switch to the next higher range
	nRangeShallowLimitPercentage	byte	The percentage of the maximum range for auto depth algorithm to switch to the next lower range
	nMinPingRateInHz	byte	The minimum ping rate in Hz required by the system
	nReserved[5]	byte	Reserved field to keep the structure aligned
GPS_QUALITY_PARAS	wQualIndicator	unsigned int16	GPS quality indicator
	wNSat	unsigned int16	Number of Satellites in use
	fHorizDilution	float	Horizontal dilution of precision
	wPosFixQualityCm	unsigned int16	Position fixed quality in cm
	wReserved	unsigned int16	Reserved field to keep the structure aligned

1.3 Data Structure

The data packets are packed with packet header and packet footer for data integrity purpose.



1.3.1 Packet Header Prefix

Size	Type	Description
2 bytes	unsigned int16	**Synchronization word, always 0x8000**
2 bytes	unsigned int16	**Synchronization word, always 0x8000**
2 bytes	unsigned int16	**Synchronization word, always 0x8000**
2 bytes	unsigned int16	**Synchronization word, always 0x8000**
2 bytes	unsigned int16	Data type, always 0x1002
2 bytes	unsigned int16	Reserved field
40 bytes	unsigned int32[10]	40 reserved bytes
4 bytes	unsigned int32	Packet body size.

1.3.2 Packet Footer

Size	Type	Description
4 bytes	unsigned int32	Packet body size.
40 bytes	unsigned int32[10]	40 reserved bytes

1.3.3 Data Integrity Checking Procedure

- Check the “Synchronization word” 0x8000,0x8000, 0x8000, 0x8000
- “Packet body size” in “Packet Header” and “Packet Footer” should be the same.

1.3.4 Packet Body Description

Packet body contains Data Header and Data Body.

Data Header

Data Header is an n bytes long field prefixed to the data body.

Field	Item size (bytes)	Data type	Description
dwBeamformed_Data_Version	4	unsigned int32	**Version of this header.** **Current version 10.** v1: TVG parameters v2: HW_Status, Sensor_Version v3: nStartElement, nEndElement v4: bHeadSensorVersion, HeadHWStatus, fInternalSensorHeading, fInternalSensorPitch, fInternalSensorRoll v5: for alignment purpose, added wReserved1, wReserved2, byReserved1, byReserved2 v6: added vessel speed over ground, wProcessingType v7: added fHeave v8: added fPRIMinRange, fPRIMaxRange v9: added fSoundSpeed_RT v10: added sRealTimeSoundSpeed and removed fSoundSpeed_RT. added sProfilingDepthTracking and sGPSQualityParas. added fAltitude and fHeightGeoidAbvEllipsoid, TimeSyncMode, bSoundSpeedSource, bTxPulseType

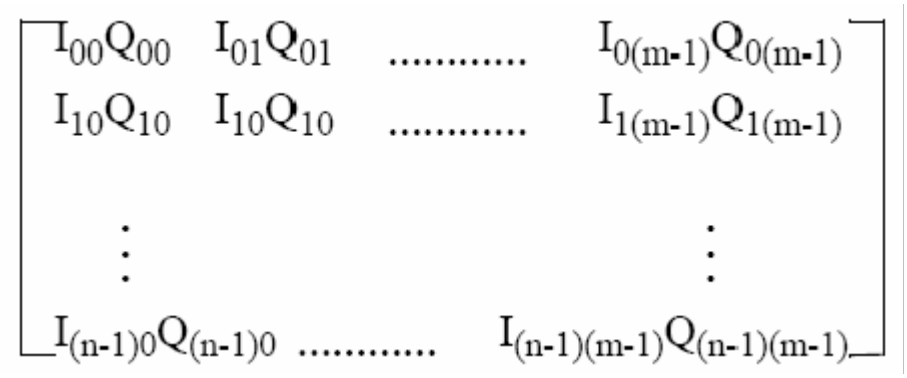
dwSonarID	4	unsigned int32	**Sonar identification**
dwSonarInfo[8]	32	unsigned int32[8]	**Sonar information such as serial number, power up configurations.**
dwTimeSec	4	unsigned int32	**Time stamp of current ping in seconds elapse since midnight (00:00:00), January 1, 1970.**
dwTimeMillisec	4	unsigned int32	**Milliseconds part of current ping.**
fSoundSpeed	4	float	**Speed of sound in m/s, applied speed of sound for image processing.**
nNumImageSample	4	unsigned int32	**Total number of image samples.**
fNearRangeMeter	4	float	**Minimum mode range in meters. No image below this range.**
fFarRangeMeter	4	float	**Maximum mode range in meters.**
fSWST	4	float	**Sampling Window Start Time in seconds**
fSWL	4	float	**Sampling Window Length in seconds**
nNumBeams	2	unsigned int16	**Total number of beams.**
wProcessingType	2	unsigned int16	**0: standard, 1: EIQ, 2: Profiling 3:Hybrid, 4:interferometry 5: Trawl 6: Profiling FFT**
fBeamList[1024]	4096	float[1024]	**A list of angles for all beams up to the number of beams specified in nNumBeams field.**
fImageSampleInterval	4	float	**Image sample interval in seconds**
nImageDestination	2	unsigned int16	**Image designation, 0: main image window, n: zoom image window n, n<=4**
bTXPulseType	1	byte	**Tx pulse type. 0: CW, 1: FM up sweep, 2: FM down sweep.**
bReserved	1	byte	**Reserved field.**
dwModeID	4	unsigned int32	**Unique mode ID, application ID**
nNumHybridPRIs	4	signed int 32	**Number of PRIs in a hybrid mode.** **1 if not in hybrid mode.**
nHybridIndex	4	signed int 32	**0 based index used in hybrid mode up to (**nNumHybridPRI -1)
nPhaseSeqLength	2	unsigned int 16	**Spatial phase sequence length**
nPhaseSeqIndex	2	unsinged int 16	**Spatial phase sequence length index 0..(nPhaseSeqLength – 1)**
nNumImages	2	unsigned int 16	**Number sub-images for one PRI**
nSubImageIndex	2	unsigned int 16	**Sub-image index 0 .. (nNumImages – 1)**
dwFrequency	4	unsigned int 32	**Sonar frequency in Hz.**

dwPulseLength	4	unsigned int 32	**Transmit pulse length in microseconds.**
dwPingNumber	4	unsigned int32	**Ping counter. Rolls back to zero if reaches 0xFFFFFFFF**
fRxFilterBW	4	float	**RX filter bandwidth in Hz**
fRxNominalResolution	4	float	**RX nominal resolution in metres**
fPulseRepFreq	4	float	**Pulse Repetition Frequency in Hz**
strAppName	128	char[128]	**Application name, null terminated. Maximum 128 characters.**
strTXPulseName	64	char[64]	**TX pulse name, null terminated. Maximum 64 characters.**
sTVGParameters	12	MUM_TVG_PARAM	**TVG Parameters: (Available if dwVersion** >=1)
fCompassHeading	4	float	**heading of current ping in decimal degrees**
fMagneticVariation	4	float	**Magnetic variation in decimal degrees**
fPitch	4	float	**Pitch in decimal degrees**
fRoll	4	float	**Roll in decimal degrees**
fDepth	4	float	**Depth in decimal meters**
fTemperature	4	Float	**Temperature in decimal Celsius**
sOffSets	28	M3_OFFSETS	**M3 deployment configuration including translational and rational offsets and mounting information**
dbLatitude	8	double	**Latitude of current ping in decimal degrees**
dbLongitude	8	double	**Longitude of current ping in decimal degrees**
fTXWST	4	float	**TX Window Start Time in seconds. Available if (**dwVersion >= 4)
bHeadSensorVersion	1	byte	**Head Sensor Version**
HeadHWStatus	1	byte	**Head hardware status:** **0: Normal** **1: High temperature warning** **2: High temperature shutdown**
bSoundSpeedSource	1	byte	**Source of the sound speed. 0: real time sensor, 1: user manually entered.**
bTimeSyncMode	1	byte	**Timer synchronization mode. 1: free running, 2: 1PPS, 4: NTP, 8: host synchronization**
fInternalSensorHeading	4	float	**Heading from internal sensor in decimal degrees**

fInternalSensorPitch	4	float	**Pitch from internal sensor in decimal degrees**
fInternalSensorRoll	4	float	**Roll from internal sensor in decimal degrees**
sAxesRotatorOffsets[3]	48	M3_ROTATOR_OFFSETS[3]	**Rotator offsets for a rotator mounted on the M3 system at Axis 1, 2 and 3.**
nStartElement	2	unsigned int 16	**Used if** nNumImages > 1. **Zero based start element of the sub array for the current sub image.**
nEndElement	2	unsigned int 16	**Used if** nNumImages > 1. **Zero based end element of the sub array for the current sub image.**
strCustomText	32	char[32]	**Custom text field**
strROVText	32	char[32]	**Custom text field 2**
fLocalTimeOffset	4	float	**Local time zone offset in decimal hours relative to UTC.**
fVesselSOG	4	float	**Vessel Speed Over Ground in knots**
fHeave	4	float	**Heave in meters.**
fPRIMinRange	4	float	**Minimum PRI range in meters.**
fPRIMaxRange	4	float	**Maximum PRI range in meters.**
fAltitude	4	float	**Altitude in decimal meters.**
fHeightGeoidAbvEllipsoid	4	float	**Ellipsoid height in meters.**
sRealTimeSoundSpeed	16	REAL_TIME_SOUND_SPEED	**A structure containing the raw, filtered, applied sound speed and the sound speed threshold, for both external sensor readings and the fixed value.**
sProfilingDepthTracking	8	PROFILING_DEPTH_TRACKING	**A structure containing parameters for profiling auto depth algorithms.**
sGPSQualityParas	12	GPS_QUALITY_PARAS	**GPS quality parameters.**
Reserved	3816	byte	**3816 reserved bytes**

Data Body

The data body contains beamformed sonar data in 32 bits floating point complex format.



Where:

n = Total number of beams, declared in field nNumBeams

m = Total number of image samples, declared in field nNumSamples

Note 1:

How to calculate image sample N position P(N) in meters:

if (dwVersion < 4)

P(N) = ((fSWST – 0.000025) + fImageSampleInterval * N) * fVelocitySound / 2

If (dwVersion >=4)

P(N) = ((fSWST – fTXWST) + fImageSampleInterval * N) * fVelocitySound / 2

Where N = 0 .. (n-1)

1.4 Ethernet Output

Protocol: TCP/IP

Socket Port: 20001

The Beamformed data packets are exported through M3 software (version >= 1.2) over TCP/IP. The third party software TCP/IP client can issue a TCP/IP socket connection at port 20001 to receive beamformed streaming socket data. No configuration in M3 is required.

2 COORDINATE SYSTEMS

This section describes the coordinate systems used in deployment configuration and related activities, including generation of 3D point clouds from ping and rotator data.

2.1 Sonar Coordinate System

The Sonar Coordinate System {S} is fixed to the face of the sonar as indicated in **Figure 1**. When the sonar is mounted in the commonly used "Forward" orientation on a vehicle, Z_S will point up, Y_S will point forward and X_S will point to starboard.

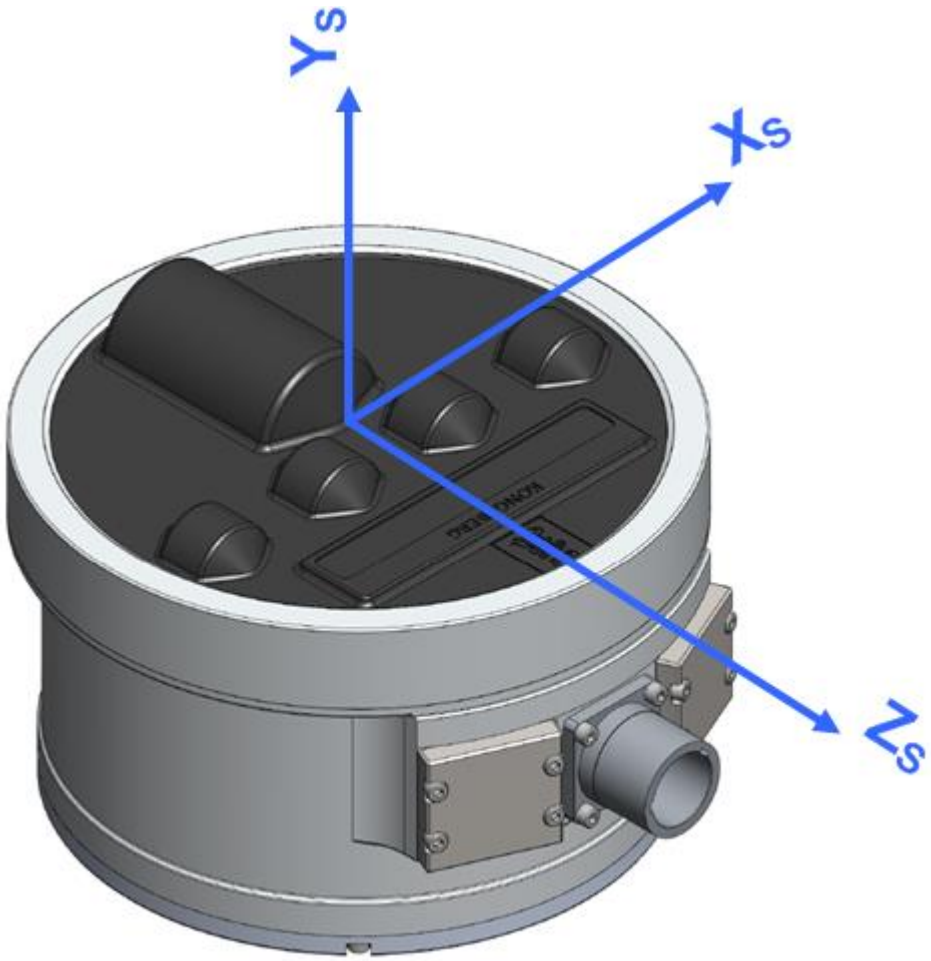


Figure 1: Sonar Coordinate System {S}

2.2 Reference Coordinate System

The Reference Coordinate System {0} is a right-handed Cartesian coordinate system (X_0, Y_0, Z_0) fixed with respect to the sonar platform (e.g., the ROV, tripod or pole-mount). By convention, Y_0 points forwards and Z_0 points upwards.

2.3 Sonar Mounting Parameters

The sonar mounting parameters describe the position of the Sonar Coordinate System {S} with respect to the Reference Coordinate System {0} as shown in **Figure 2**. There are three translational parameters (X offset, Y offset, Z offset) and three rotational parameters (pitch, roll, yaw), which are applied in a translate-then-rotate fashion.

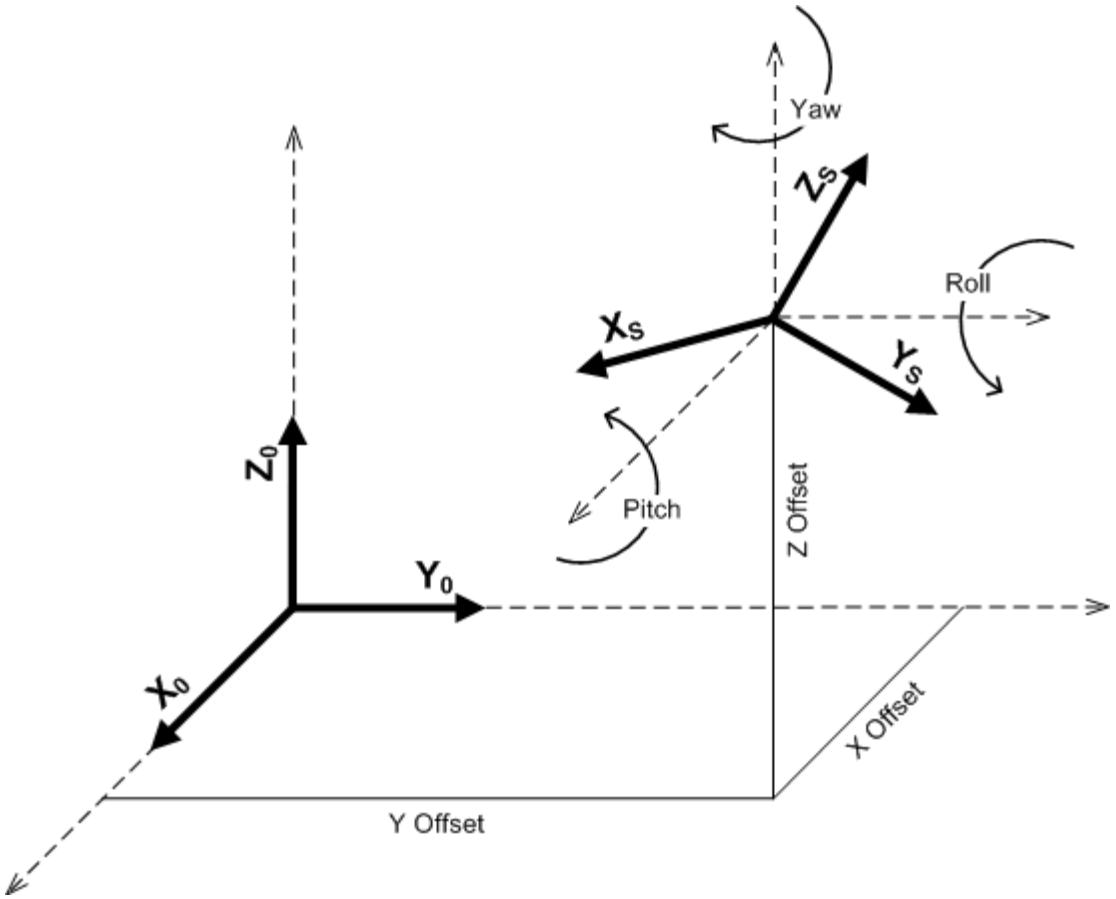


Figure 2: Sonar Mounting Parameters

2.4 Coordinate Transformations

Coordinate transformations are defined by translation and rotation parameters (angles).

The following matrix equation describes the transformation of a point P in {S} coordinates to point P in {0} coordinates:

$$ {}^{O}P \;=\; {}^{O}_{S}R\,{}^{S}P \;+\; {}^{O}P_{SORG} $$

where:

${}^{O}P$ is point P with respect to {0},

${}^{S}P$ is point P with respect to {S},

${}^{O}_{S}R$ is the rotation of {S} with respect to {0}, and

${}^{O}P_{SORG}$ is the origin of {S} with respect to {0}.

In terms of the transformation parameters, ${}^{O}P_{SORG}$ is a 3 X 1 vector containing the translation parameters, and ${}^{O}_{S}R$ is a 3 X 3 rotation matrix that generated from the three rotation parameters.

The matrix equation for the inverse transformation is given by

$$ {}^{S}P \;=\; {}^{S}_{O}R\left({}^{O}P \;-\; {}^{O}P_{SORG}\right) $$

Rotation is defined in terms of *X-Y-Z fixed angles*, where the word "fixed" refers to the fact that the rotations are specified about the fixed (non-moving) reference frame. In *X-Y-Z fixed angles*, the rotations are carried out in the following order:

1. Rotate {S} about $\hat{X}_O$ by γ
2. Rotate {S} about $\hat{Y}_O$ by β
3. Rotate {S} about $\hat{Z}_O$ by α

Rotation direction is clockwise when looking in the direction of it respective axis (right-hand rule). For the sonar mounting rotational parameters: pitch = γ; roll = β; and yaw = $-\alpha$.

The {S} to {0} rotation matrix is defined

$$ {}^{O}_{S}R \;=\; R_Z(\alpha)\,R_Y(\beta)\,R_X(\gamma) = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} $$

where:

$$r_{11} = \cos\alpha \cos\beta$$
$$r_{21} = \sin\alpha \cos\beta$$
$$r_{31} = -\sin\beta$$
$$r_{12} = \cos\alpha \sin\beta \sin\gamma - \sin\alpha \cos\gamma$$
$$r_{22} = \sin\alpha \sin\beta \sin\gamma + \cos\alpha \cos\gamma$$
$$r_{32} = \cos\beta \sin\gamma$$
$$r_{13} = \cos\alpha \sin\beta \cos\gamma + \sin\alpha \sin\gamma$$
$$r_{23} = \sin\alpha \sin\beta \cos\gamma - \cos\alpha \sin\gamma$$
$$r_{33} = \cos\beta \cos\gamma$$

Section 2.6 contains C++ sample code showing coordinate transformation implementation.

2.4.1 Comparing Transformations

Two transformations may be compared by calculating the norm of the difference between the two points resulting from transforming a common input point separately by each of the transformations.

$${}^{A}P = {}^{O}_{A}T\,{}^{O}P$$

$${}^{B}P = {}^{O}_{B}T\,{}^{O}P$$

$$d = \left\| {}^{A}P - {}^{B}P \right\|$$

The norm d gives a measure of "distance" between transformations ${}^{O}_{A}T$ and ${}^{O}_{B}T$ for the common input point ${}^{O}P$. Two or more separate input points are required to give a proper measure of "distance" between transformations; a single point will indicate zero distance for an infinite number of transformation pairs. Note, this matrix notation combines the rotation matrix and translation vector of the transformation.

2.5 Rotator Axes and Offsets

Up to three rotator axes are supported in the deployment configuration. Each rotator axis is defined relative to the sonar coordinate system (X_S, Y_S, Z_S) in term of System Axes:

1. System Axis 1 is parallel to X_S when rotator angles on axes 2 and 3 are zero
2. System Axis 2 is parallel to Y_S when rotator angles on axes 1 and 3 are zero
3. System Axis 3 is parallel to Z_S when rotator angles on axes 1 and 2 are zero

See **Figure 1** for details of the sonar coordinate system (X_S, Y_S, Z_S).

Each rotator axis is defined relative to the sonar coordinate system in terms of rotator offsets A, B and R as shown in **Figure 3**, **Figure 4** and **Figure 5** for each of the three system axes. **Figure 6** and Table 1 describe coordinate transformations in terms of the rotator offsets and angles (see Section 2.6 for a corresponding C++ code example).

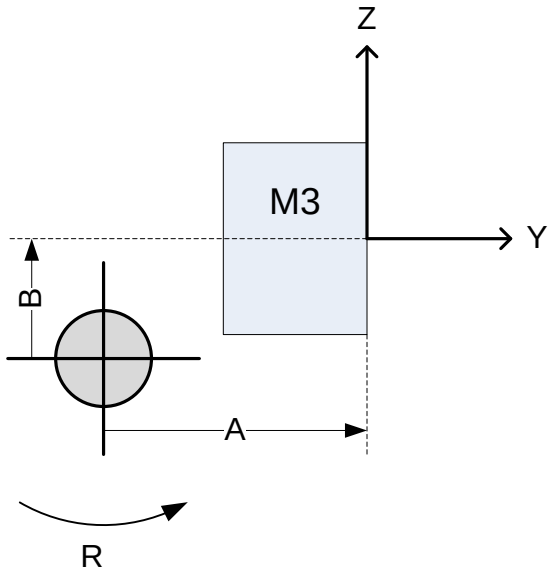


Figure 3: Rotator Offsets for System Axis 1

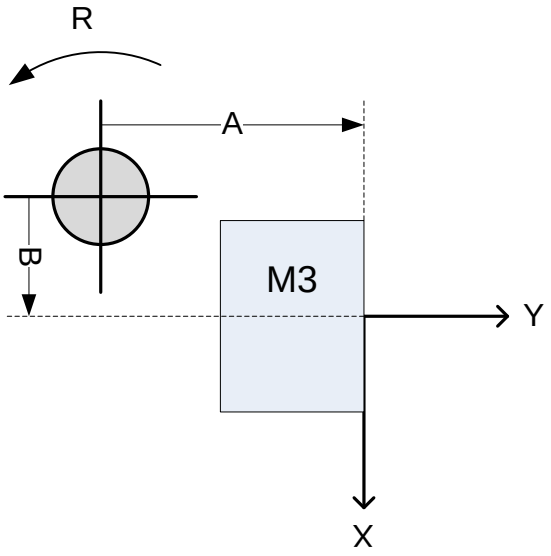


Figure 4: Rotator Offsets for System Axis 2

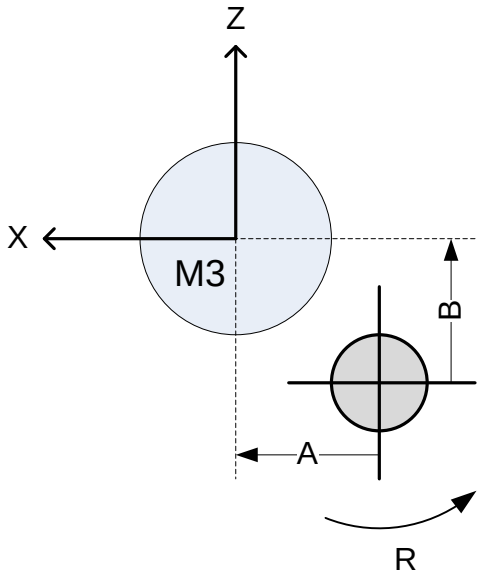


Figure 5: Rotator Offsets for System Axis 3

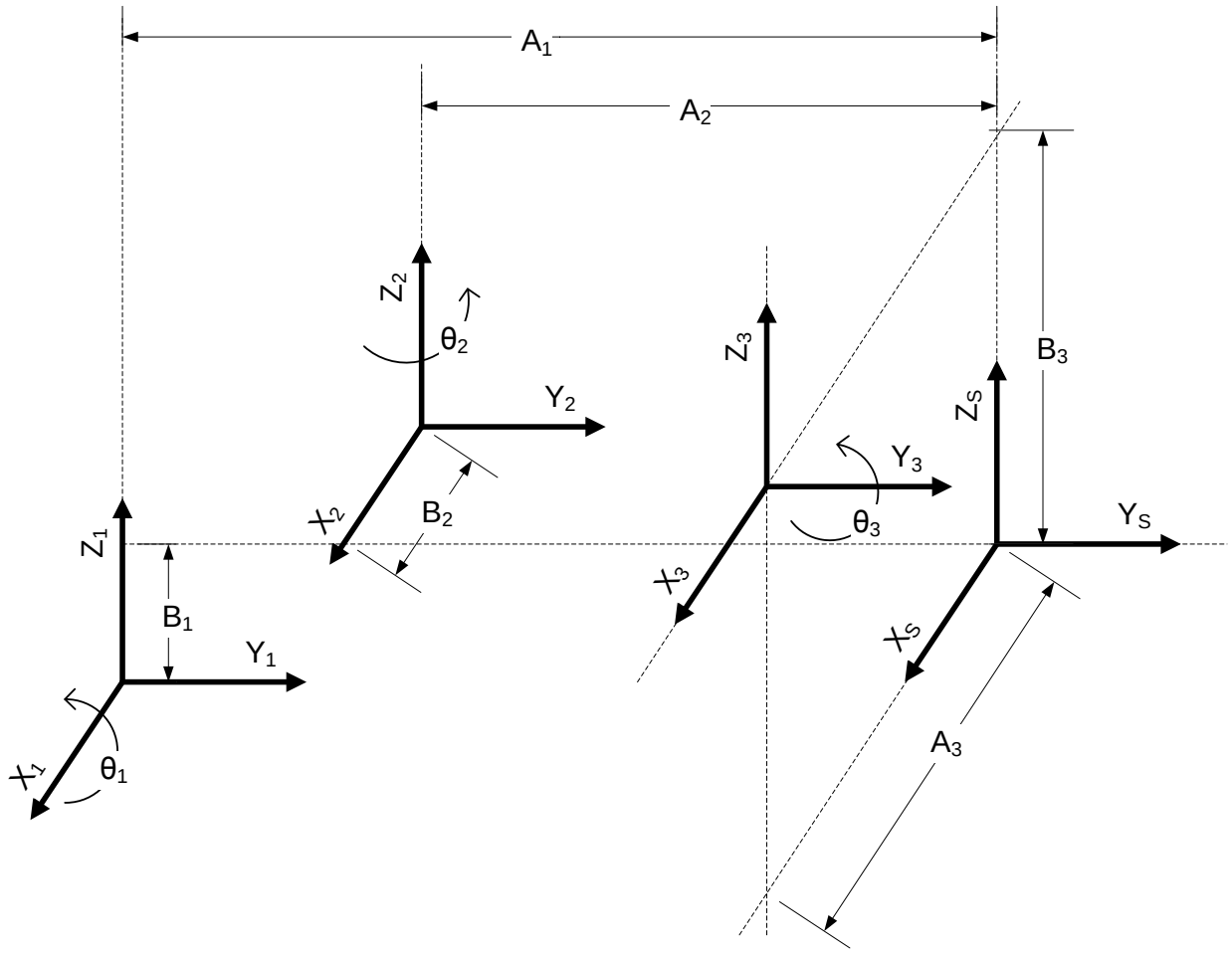


Figure 6: Rotator Offsets and Angles

Table 1: Coordinate Transformations in terms of Rotator Offsets

	${}^{S}_{1}T$	${}^{1}_{2}T$	${}^{2}_{3}T$	${}^{3}_{S}T$	${}^{1}_{S}T_{0}$
Δx	$-B_2$	0	$B_2 + A_3$	$-A_3$	B_2
Δy	$-A_1$	$A_1 - A_2$	A_2	0	A_1
Δz	$-B_1$	B_1	B_3	$-B_3$	B_1
γ	θ_1	0	0	0	0
β	0	0	θ_3	0	0
α	0	θ_2	0	0	0

2.6 Sample Code for Coordinate Transformation

2.6.1 CoordSys Header

```
// CoordSys.h

class CoordSys
{
public:

  CoordSys(double x,
           double y,
           double z,
           double gamma,
           double beta,
           double alpha);

  ~CoordSys(void);

  void Concat(const CoordSys& B);
  void Transform(double& x, double& y, double& z) const;
  void Matrix( GLfloat matrix[ 16 ] );

  //-------------------------------------------------------------------------
  // Norm -- designed for use in comparing transforms. The norm returned by
  // this function is a measure of "distance" between transforms in the
  // direction specified by the input vector.
  //
  // Returns the norm ||r_a - r_b||, where
  //
  //   r_a is the transformation of vector r by this transform
  //   r_b is the transformation of vector r by transform B
  //   r is the input vector specified by (x,y,z)
  //
  //-------------------------------------------------------------------------
  //
```

```
  double Norm(const CoordSys& B, const double x, const double y, const
double z) const;

private:
  CoordSys();



private:
  double P_org[3];
  double R_xyz[3][3];

};
```

2.6.2 CoordSys Source

```
// CoordSys.cpp

#include "CoordSys.h"
#include <cmath>

#define DEG2RAD (double)0.017453292519943

CoordSys::CoordSys(double x,
                   double y,
                   double z,
                   double gamma,
                   double beta,
                   double alpha)
{
    P_org[0] = x;
    P_org[1] = y;
    P_org[2] = z;

    double sin_a = sin(alpha*DEG2RAD);
    double sin_b = sin(beta*DEG2RAD);
    double sin_g = sin(gamma*DEG2RAD);
```

```
    double cos_a = cos(alpha*DEG2RAD);
    double cos_b = cos(beta*DEG2RAD);
    double cos_g = cos(gamma*DEG2RAD);

    R_xyz[0][0] = cos_a*cos_b;
    R_xyz[1][0] = sin_a*cos_b;
    R_xyz[2][0] = -sin_b;

    R_xyz[0][1] = cos_a*sin_b*sin_g - sin_a*cos_g;
    R_xyz[1][1] = sin_a*sin_b*sin_g + cos_a*cos_g;
    R_xyz[2][1] = cos_b*sin_g;

    R_xyz[0][2] = cos_a*sin_b*cos_g + sin_a*sin_g;
    R_xyz[1][2] = sin_a*sin_b*cos_g - cos_a*sin_g;
    R_xyz[2][2] = cos_b*cos_g;
}

CoordSys::~CoordSys(void)
{
}

void CoordSys::Concat(const CoordSys& B)
{
    int i, j = 0;

    // Make local copy of the rotation matrix
    double R[3][3];

    for (i = 0; i < 3; i++)
    {
        for (j = 0; j < 3; j++)
        {
            R[i][j] = R_xyz[i][j];
        }
    }

    // Loop over rows
```

```
    for (i = 0; i < 3; i++)
    {
        P_org[i] =
            R[i][0]*B.P_org[0] +
            R[i][1]*B.P_org[1] +
            R[i][2]*B.P_org[2] +
            P_org[i];

        for (j = 0; j < 3; j++)
        {
            R_xyz[i][j] =
                R[i][0]*B.R_xyz[0][j] +
                R[i][1]*B.R_xyz[1][j] +
                R[i][2]*B.R_xyz[2][j];
        }
    }
}

void CoordSys::Transform(double& x, double& y, double& z) const
{
    double t[3];

    for (int i = 0; i < 3; i++)
    {
        t[i] = R_xyz[i][0]*x + R_xyz[i][1]*y + R_xyz[i][2]*z + P_org[i];
    }

    x = t[0];
    y = t[1];
    z = t[2];
}

double CoordSys::Norm(const CoordSys& B, const double x, const double y,
const double z) const
{
  double a_x = x;
  double a_y = y;
  double a_z = z;
```

```
  double b_x = x;
  double b_y = y;
  double b_z = z;

  this->Transform(a_x, a_y, a_z);
  B.Transform(b_x, b_y, b_z);

  double distance = sqrt( (a_x - b_x)*(a_x - b_x) +
                          (a_y - b_y)*(a_y - b_y) +
                          (a_z - b_z)*(a_z - b_z) );

  return distance;

}

void CoordSys::Matrix( GLfloat matrix[ 16 ] )
{
   #define M( x, y )  matrix[ x + y * 4 ]

   M( 0, 0 ) = (GLfloat)R_xyz[0][0];
   M( 1, 0 ) = (GLfloat)R_xyz[1][0];
   M( 2, 0 ) = (GLfloat)R_xyz[2][0];
   M( 3, 0 ) = (GLfloat)0.0;

   M( 0, 1 ) = (GLfloat)R_xyz[0][1];
   M( 1, 1 ) = (GLfloat)R_xyz[1][1];
   M( 2, 1 ) = (GLfloat)R_xyz[2][1];
   M( 3, 1 ) = (GLfloat)0.0;

   M( 0, 2 ) = (GLfloat)R_xyz[0][2];
   M( 1, 2 ) = (GLfloat)R_xyz[1][2];
   M( 2, 2 ) = (GLfloat)R_xyz[2][2];
   M( 3, 2 ) = (GLfloat)0.0;

   M( 0, 3 ) = (GLfloat)0.0;
   M( 1, 3 ) = (GLfloat)0.0;
```

```
  M( 2, 3 ) = (GLfloat)0.0;
  M( 3, 3 ) = (GLfloat)1.0;
}
```

2.6.3 Sample Usage of CoordSys

```
#include "CoordSys.h"

  // ...

  CoordSys T_RS(xOffset, yOffset, zOffset,  pitch,   roll,   -yaw);
  CoordSys T_S1(    -b2,     -a1,     -b1, theta1,    0.0,    0.0);
  CoordSys T_12(    0.0, a1 - a2,      b1,    0.0,    0.0, theta2);
  CoordSys T_23(b2 + a3,      a2,      b3,    0.0, theta3,    0.0);
  CoordSys T_3S(    -a3,     0.0,     -b3,    0.0,    0.0,    0.0);
  CoordSys T_1S(     b2,      a1,      b1,    0.0,    0.0,    0.0);

  T_RS.Concat(T_S1);
  T_RS.Concat(T_12);
  T_RS.Concat(T_23);
  T_RS.Concat(T_3S);
  T_RS.Concat(T_1S);

  for (int i = 0; i < numPoints; ii++)
  {
    T_RS.Transform(x[i], y[i], z[i]);
  }
  // ...
```