Dynamic orientation calibration

M. de Jong

January 17, 2024

Abstract

The dynamic orientation calibration of the detector using the Jpp software framework is presented. The procedure is based on polynomial interpolations of the compass data. The compasses are aligned using a fit of a model to the data from the same string.

1 Introduction

The KM3NeT research infrastructure comprises several large 3D-arrays of optical modules deployed in the deep waters of the Mediterranean Sea. These optical modules consist of a glass sphere which houses the photo-multiplier tubes (PMTs) and the readout electronics that are used for the detection of neutrinos. The optical modules are arranged on strings which are anchored on the seabed and held vertically by the buoyancy of the glass spheres, further supported by a buoy on top. To achieve the envisaged angular resolution of the detected neutrinos, the orientations of the optical modules should be measured with an accuracy of about $3\ \mathrm{deg}$. Due to varying sea currents, the modules rotate in the sea water. To accurately follow the rotations of the optical modules, their orientations should be measured every 10 minutes or so. The repeated measurements of the orientations of the optical modules are referred to as "dynamic orientation calibration".

For the dynamic orientation calibration, an attitude and heading reference system (AHRS) is used, conveniently referred to as "compass". To this end, a commercial device (LSM) is mounted in each optical module and regularly read out. Earlier optical modules contain a custom device (AHRS).
The compass data are continually taken during normal operation of the detector (currently every 10 seconds) and stored in the central database in table "ahrs". The list of columns of this table includes:

name	data type
DETID	int
RUN	int
UNIXTIME	long long int
DUID	int
FLOORID	int
CLBUPI	JUPI_t
DOMID	int
AHRS_A(0-2)	double
AHRS_G(0-2)	double
AHRS_H(0-2)	double

In this, DETID corresponds to the identifier of the detector (SERIALNUMBER in table "detectors"), RUN to the number of the data taking run, DOMID to the identifier of the module which relates to the compass, and UNIXTIME to the time at which the data were recorded (s). The data in columns DUID, FLOORID and CLBUPI are not used. The notation (0-2) corresponds to multiple columns of which the name includes the

character in the indicated sequence. The data type JUPI_t is a custom class that is used to parse the universal product identifier (UPI) of a component in the detector. The measured values of the accelerometer (AHRS_A(0-1)) and the magnetic field sensor (AHRS_H(0-1)) are used to determine the yaw, pitch and roll of the optical module.

To determine the yaw, pitch and roll of the optical module, the raw data need to be calibrated first. The calibration data are stored in the central database in table "ahrscalib". The list of columns of this table includes:

name	data type
TESTOPID	std::string
TESTNAME	std::string
TESTSTART	JDatim_t
TESTEND	JDatim_t
PRODUCTTESTID	std::string
SERIALNUMBER	int
REVTIMEORDER	int
OPERATIONREPORT	std::string
FIRMWARE_VERSION	std::string
KALMAN_FILTER	std::string
MAG_DECL	double
ACC_GAIN_(X-Z)	double
ACC_OFFSET_(X-Z)	double
GYRO_GAIN_(X-Z)	double
MAG_(X-Z)MIN	double
MAG_(X-Z)MAX	double
MAG_ROT(X-Z)(X-Z)	double
GYRO_ROT(X-Z)(X-Z)	double
ACC_ROT(X-Z)(X-Z)	double

The SERIALNUMBER of the device is used to relate the calibration data to the raw data (see class JAHRSCalibration_t). For this, the data in table "detectorintegration" are used (see class JDetectorIntegration_t). The notation (X-Y) corresponds to multiple columns of which the name includes the character in the indicated sequence. The data type JDatim_t is a custom class that is used to parse the date and time of the start and end of the test. There may be several entries in this table for the same device (i.e. same SERIALNUMBER). The entries are therefore sorted and the "best" entry selected. The sorting criteria are implemented in class JAHRSCalibrationComparator. The selection also involves acceptance criteria of the calibration data. These are implemented in class JAHRSCalibrationValidity.

The conversion of raw data to yaw, pitch and roll values is implemented in class JCompass. In this class, the sign conventions of the various angles are defined as well as the correction for the magnetic declination and the meridian convergence angle. The auxiliary script get_convergence_angle.(py|sh) can be used to determine the meridian convergence angle for a given site or detector, respectively. The values for the ARCA and ORCA sites are defined in include file JCompassSupportkit.hh. The auxiliary script makedeclinationtable.(py|sh) can be used to download the magnetic declination data from the NOAA website[2]. Additional data should be imported in class JARCAMagneticDeclination and JORCAMagneticDeclination when the actual time exceeds that of the currently available data. Finally, the JCompass class also contains the member method getQuaternion which returns the values of the quaternion corresponding to the actual rotation of the optical module. A quaternion constitutes a four vector which describes a rotation around a given axis $\hat{u}$ with an angle θ: $Q \equiv \left(\frac{\cos\theta}{2}, \frac{\sin\theta}{2}\hat{u}\right)$. Quaternions are

used for the dynamic orientation calibration.

2 Calibration

After the calibration of the raw data (see above), a second calibration is made to align the compasses in the optical modules in the same string. For this, a model is fitted to the quaternion data of a single string covering a predefined period (typically 10 minutes). The following polynomial function is used to model the swing and twist of a string.

$$Q = Q_0 Q_1^{z_i} \tag{1}$$

where z_i corresponds to the height of floor i (unit m) according to the static detector geometry. The overall swing and twist of the string and the height dependence thereof are described by the quaternions Q_0 and Q_1, respectively. The χ^2 of the fit is defined by the square of the geodesic angle between the quaternion data and the model, normalised to some assumed resolution (typically $1\ \mathrm{deg}$). Multiple fits should be applied to a sufficient amount of data. The average residual (i.e. average quaternion between the measured quaternion and modeled quaternion) per optical module then constitutes the alignment of the compass. These averages are stored in the detector file conform format V4. This is referred to as the "pre-calibration" of the compasses. The whole procedure can be executed with a single script, namely `JCompass.sh`. This script takes a detector file and a list of run numbers (and/or ranges thereof) as arguments. The detector file will then accordingly be updated so that it can be used in future applications. For a standard data taking run with the current ORCA detector (`OID`="D_ORCA006" or `SERIALNUMBER`=49), the time to process the data and repeat the fits amounts to about $2\ \mathrm{s}$ per data taking run and per string.

For each detector, a complete pre-calibration procedure is implemented in a designated script called `pre-calibration_XXXXXXXX.sh`, where `XXXXXXXX` corresponds to the identifier of the detector (i.e. `OID` in table "detectors"). These scripts are available in directory `$JPP_DIR/examples/JCompass/`. In each script, a list of suitable data taking runs is specified. The selection of runs is based on the quality of the compass data and the environmental conditions (e.g. low sea currents). In addition, none of the optical modules should have a temporal rotation beyond that modelled because otherwise, this rotation would be attributed to a misalignment and stoted as such in the detector file.

3 Application

Following the calibration procedure outlined above, the available data can now be processed for the dynamic orientation calibration. To efficiently use the data from the database, the data are downloaded, converted to ROOT format and written to disk using the general purpose application `JConvertDB`. The locally stored data are subsequently processed with application `JBallarat`[1]. The auxiliary script `JBallarat.sh` does both steps in one go for a given list of run numbers (and/or ranges thereof). To eliminate possible outliers, the data are organised per module and sorted in time. All elements are removed which deviate more than a predefined angle (typically $5\ \mathrm{deg}$) from a local interpolation of the surrounding data. The current interpolation is based on Legendre polynomials [1]. The number of points and the degree of the polynomial are fixed to 20 and 1, respectively. The filtered data are written to disk in a ROOT formatted file. For a standard data taking run with the current ORCA detector (`OID`="D_ORCA006" or `SERIALNUMBER`=49), the time to process the data amounts to about 3 seconds per data taking run and the data volume to about $6\ \mathrm{MB}$.

The output of application `JBallarat` can subsequently be loaded in memory using class `JDynamics`. In this,

[1] Ballarat is a previous gold-miners town in Victoria, Australia.

Legendre polynomial interpolations are used to provide the dynamic orientation calibration of each optical module at a specified time. To efficiently evaluate the results, the previous rotations are stored memory. Each orientation is then defined by the product of the actual rotation and the inverse of its previous rotation. As a result, the time dependent orientation of an optical module evolves as a sequence of relatively small rotations.

4 Summary

A dynamic orientation calibration of the detector is implemented in the Jpp software framework. The overall procedure is fast and involves little overhead. More (technical) documentation is available in Doxygen format at `https://common.pages.km3net.de/jpp/` (see `Namespaces` → `JCOMPASS`).

References

[1] See e.g. `https://en.wikipedia.org/wiki/Legendre_polynomials`.

[2] See e.g. `https://www.ngdc.noaa.gov/geomag/calculators/magcalc.shtml`.