Anomaly Detection Toolbox - Embedded Source Code Reference Manual

Rev. 0.1 - 28 April 2017

Generated by Doxygen 1.8.13

Contents

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

Data Structure Documentation

4.1 AccelBuffer Struct Reference

accelerometer measurement buffer

```
#include <precisionAccelerometer.h>
```

Data Fields

- float fGsStored [MAX_ACCEL_CAL_ORIENTATIONS][3]
  uncalibrated accelerometer measurements (g)
- float fSumGs [3]
  averaging sum for current storage location
- int16_t iStoreCounter
  number of remaining iterations at FUSION_HZ to average measurement
- int16_t iStoreLocation
  -1 for none, 0 to 11 for the 12 storage locations
- int16_t iStoreFlags
  denotes which measurements are present

4.1.1 Detailed Description

accelerometer measurement buffer

Definition at line 43 of file precisionAccelerometer.h.

The documentation for this struct was generated from the following file:

- precisionAccelerometer.h

4.2 AccelCalibration Struct Reference

precision accelerometer calibration structure

```
#include <precisionAccelerometer.h>
```

Data Fields

- float fV [3]

  offset vector (g)
- float finvW [3][3]

  inverse gain matrix
- float fR0 [3][3]

  forward rotation matrix for measurement 0
- float fmatA [10][10]

  scratch 10x10 matrix used by calibration algorithms
- float fmatB [10][10]

  scratch 10x10 matrix used by calibration algorithms
- float fvecA [10]

  scratch 10x1 vector used by calibration algorithms
- float fvecB [4]

  scratch 4x1 vector used by calibration algorithms
- float fA [3][3]

  ellipsoid matrix A
- float finvA [3][3]

  inverse of the ellipsoid matrix A

4.2.1 Detailed Description

precision accelerometer calibration structure

Definition at line 53 of file precisionAccelerometer.h.

The documentation for this struct was generated from the following file:

- precisionAccelerometer.h

4.3 AccelSensor Struct Reference

The AccelSensor structure stores raw and processed measurements for a 3-axis accelerometer.

```
#include <anomaly_detection.h>
```

Data Fields

- uint8_t iWhoAmI

  sensor whoami
- bool isEnabled

  true if the device is sampling
- uint8_t iFIFOCount

  number of measurements read from FIFO
- uint16_t iFIFOExceeded

  Number of samples received in excess of software FIFO size.
- float fFloatPerCount

  g per count
- float fSum [4]

  sum of all measurements in this epoch
- float fSum2 [4]

  sum of squares over all measurements in this epoch
- float features [NUM_FEATURES][4]

  rows are mean, variance, SF, kurtosis and mean crossing rate
- float fFIFO [ACCEL_FIFO_SIZE][4]

  FIFO measurements.

4.3.1 Detailed Description

The AccelSensor structure stores raw and processed measurements for a 3-axis accelerometer.

The AccelSensor structure stores raw and processed measurements, as well as metadata for a single 3-axis accelerometer. This structure is normally "fed" by the sensor driver and "consumed" by the fusion routines.

Definition at line 258 of file anomaly_detection.h.

The documentation for this struct was generated from the following file:

- anomaly_detection.h

4.4 Cache Class Reference

Public Member Functions

- **Cache** (int l, long int size)
- int **get_data** (const int index, Qfloat $**$data, int len)
- void **swap_index** (int i, int j)

4.4.1 Detailed Description

Definition at line 51 of file svm.cpp.

The documentation for this class was generated from the following file:

- svm.cpp

4.5 ControlSubsystem Struct Reference

he ControlSubsystem encapsulates command and data streaming functions.

```
#include <control.h>
```

Data Fields

- volatile uint8_t flagOne
  1st boolean flag
- bool StreamEnable
  Mode to control streaming.
- bool TrainRun
  Mode for Start.
- bool CLR
  Command: stop and clear existing model states.
- bool Stop
  Command: stop model operation.
- bool Start
  Command: start model operation (controlled via modes)
- bool Delete
  Command: Delete specific model.
- bool DeleteAll
  Command: Delete ALL models.

4.5.1 Detailed Description

he ControlSubsystem encapsulates command and data streaming functions.

The ControlSubsystem encapsulates command and data streaming functions for the library. A C++-like typedef structure which includes executable methods for the subsystem is defined here.

Definition at line 56 of file control.h.

The documentation for this struct was generated from the following file:

- control.h

4.6 decision_function Struct Reference

Data Fields

- double ∗ **alpha**
- double **rho**

4.6.1 Detailed Description

Definition at line 1067 of file svm.cpp.

The documentation for this struct was generated from the following file:

- svm.cpp

4.7 FEATURE_LIST Struct Reference

Data Fields

- bool **in_use**
- sensor_t **sensor**
- axis_t **axis**
- feature_t **feature**

4.7.1 Detailed Description

Definition at line 36 of file feature_list.c.

The documentation for this struct was generated from the following file:

- feature_list.c

4.8 FeatureCollection Struct Reference

Feature Collection structure contains the poinsters of FeatureInstance structures.

```
#include <machineLearning_subsystem.h>
```

Data Fields

- struct FeatureInstance ∗ pFeatureInstance [MAX_FEATURE_INSTANCES]
  pointers of feature instances

4.8.1 Detailed Description

Feature Collection structure contains the poinsters of FeatureInstance structures.

Potentially, control parameters for a collection of Feature Instances may be added here.

Definition at line 77 of file machineLearning_subsystem.h.

The documentation for this struct was generated from the following file:

- machineLearning_subsystem.h

4.9 FeatureInstance Struct Reference

Feature Instance structure contains the buffers for feature samples and other information.

```
#include <machineLearning_subsystem.h>
```

Data Fields

- feature_t feature_type
  Feature type defined in feature_t.
- sensor_t sensor_type
  Sensor type defined in sensor_t.
- axis_t sensor_axis
  Sensor axis defined in axis_t.
- int16_t iBufferCount
  Count feature samples when stored in fBuffer.
- bool iBufferExceeded
  True if fBuffer is filled.
- bool isEnabled
  True if this FeatureInstance structure was enabled.
- bool isNormalized
  True if this FeatureInstance was normalized (assumed zscore, i.e. standardization)
- bool doNormalization
  True if feature normalization is reqested.
- float fBufferMean
  Sample mean to compute zscore, within a moving window whose size is defined by featureBufferSize in ModelInstance structure.
- float fBufferStd
  Sample std to compute zscore, within a moving window whose size is defined by featureBufferSize in ModelInstance structure.
- float fBufferNormalized [MAX_FEATURE_SAMPLES]
  A buffer array, normalized from fBuffer.
- float fBuffer [MAX_FEATURE_SAMPLES]
  A buffer that contains un-normalized features from raw sensor data.

4.9.1 Detailed Description

Feature Instance structure contains the buffers for feature samples and other information.

Feature type, sensor type, and sensor axis are used to identify feature instance. fBufferNormalized is the buffer for normalized features (zscore) from fBuffer, fBufferMean, and fBufferStd. The other variables in FeatureInstance structure are control variables.

Definition at line 59 of file machineLearning_subsystem.h.

The documentation for this struct was generated from the following file:

- machineLearning_subsystem.h

4.10 FifoSensor Union Reference

The FifoSensor union allows us to use common pointers for Accel, Mag & Gyro logical sensor structures.

```
#include <anomaly_detection.h>
```

Data Fields

- struct GyroSensor **Gyro**
- struct MagSensor **Mag**
- struct AccelSensor **Accel**

4.10.1 Detailed Description

The FifoSensor union allows us to use common pointers for Accel, Mag & Gyro logical sensor structures.

Common elements include: iWhoAmI, isEnabled, iFIFOCount, iFIFOExceeded and the FIFO itself.

Definition at line 315 of file anomaly_detection.h.

The documentation for this union was generated from the following file:

- anomaly_detection.h

4.11 GaussComponent Struct Reference

A Gaussian component structure in the mixture model.

```
#include <machineLearning_subsystem.h>
```

Data Fields

- float N

  Soft number of samples assigned to this Gaussian component.
- float pi

  Mixing probability.
- float cnst

  1/sqrt((2PI)^dim ∗ determinant) in log scale.
- float muMeans [MAX_FEATURE_DIMENSION]

  Mean vector.
- float rCovariance [MAX_FEATURE_DIMENSION][MAX_FEATURE_DIMENSION]

  Covariance matrix.
- float rInvCovariance [MAX_FEATURE_DIMENSION][MAX_FEATURE_DIMENSION]

  Inverse covariance matrix.

4.11.1 Detailed Description

A Gaussian component structure in the mixture model.

This structure contains the computational information as well as the parameter set of a single Gaussian component in the mixture model. GaussCompoentn is included in GmmModel structure.

Definition at line 85 of file machineLearning_subsystem.h.

The documentation for this struct was generated from the following file:

- machineLearning_subsystem.h

4.12 Globals Struct Reference

The top level fusion structure.

```
#include <anomaly_detection.h>
```

Data Fields

SubsystemPointers

The Status and Control subsystems can be used as-is, or completely replaced with alternate implementations, as long as those implementations provide the same interfaces defined in control.h and status.h.

- struct ControlSubsystem ∗ **pControlSubsystem**
- struct StatusSubsystem ∗ **pStatusSubsystem**
- struct FeatureCollection ∗ pFeatureCollection

  pointer for feature operation object
- struct ModelCollection ∗ pModelCollection

  pointer model operation object
- uint8_t controlCommandforML

  machine learning control commands received from GUI: e.g., start training, download, etc.
- uint8_t **current_model_id**

MiscFields

- uint32_t iFlags

  a bit-field of sensors and algorithms used
- struct PhysicalSensor ∗ pSensors

  a linked list of physical sensors
- volatile uint8_t iPerturbation

  test perturbation to be applied
- int32_t loopcounter

  counter incrementing each iteration of sensor fusion (typically 25Hz)
- int32_t systick_I2C

  systick counter to benchmark I2C reads
- int32_t systick_Spare

  systick counter for counts spare waiting for timing interrupt

FunctionPointers

Function pointers (the SF library external interface)

- installSensor_t ∗ installSensor

    function for installing a new sensor into t
- initializeFusionEngine_t ∗ initializeAD

    set sensor fusion structures to initial values
- readSensors_t ∗ readSensors

    read all physical sensors
- runFusion_t ∗ runAD

    run the fusion routines
- clearFIFOs_t ∗ clearFIFOs

    clear sensor FIFOs
- setStatus_t ∗ setStatus

    change status indicator immediately
- setStatus_t ∗ queueStatus

    queue status change for next regular interval
- updateStatus_t ∗ updateStatus

    status=next status
- updateStatus_t ∗ testStatus

    increment to next enumerated status value (test only)

4.12.1 Detailed Description

The top level fusion structure.

The top level fusion structure grows/shrinks based upon flag definitions contained in build.h. These same flags will populate the .iFlags field for run-time access.

Definition at line 326 of file anomaly_detection.h.

The documentation for this struct was generated from the following file:

- anomaly_detection.h

4.13 GmmModel Struct Reference

A Gaussian mixture model (GMM) structure.

```
#include <machineLearning_subsystem.h>
```

Data Fields

- bool **initialized**
- uint8_t opt_num_components

    The optimal number of GaussComponent in the mixture model. It is set after trained.
- uint8_t init_maxGaussComponents

    The allowed max number of GaussComponent for training.
- int8_t nComponents

    The number of GaussComponent used during training time.
- float Rmin

    For regularization of covariance matrix. Singularity may be avoided.
- float pProb [MAX_FEATURE_SAMPLES][MAX_GAUSSIAN_COMPONENTS]

    The probability that a feature sample belongs to a mixture component. For example, pProb[i][j] represents the probability for feature sample i to j component.
- GaussComponent gComponents [MAX_GAUSSIAN_COMPONENTS]

    Collection of Gaussian components used for the mixture model.

4.13.1 Detailed Description

A Gaussian mixture model (GMM) structure.

GMM parameters as well as trained GMM by expectation-maximization (EM) algorithm. These are linked to Model↩
Instance.

Definition at line 98 of file machineLearning_subsystem.h.

The documentation for this struct was generated from the following file:

- machineLearning_subsystem.h

4.14 GyroSensor Struct Reference

The GyroSensor structure stores raw and processed measurements for a 3-axis gyroscope.

```
#include <anomaly_detection.h>
```

Data Fields

- uint8_t iWhoAmI

  sensor whoami
- bool isEnabled

  true if the device is sampling
- uint8_t iFIFOCount

  number of measurements read from FIFO
- uint16_t iFIFOExceeded

  Number of samples received in excess of software FIFO size.
- float fFloatPerCount

  deg/s per count
- float fSum [4]

  sum of all measurements in this epoch
- float fSum2 [4]

  sum of squares over all measurements in this epoch
- float features [NUM_FEATURES][4]

  rows are mean, variance, SF, kurtosis and mean crossing rate
- float fFIFO [GYRO_FIFO_SIZE][4]

  FIFO measurements (counts)

4.14.1 Detailed Description

The GyroSensor structure stores raw and processed measurements for a 3-axis gyroscope.

The GyroSensor structure stores raw and processed measurements, as well as metadata for a single 3-axis gyroscope. This structure is normally "fed" by the sensor driver and "consumed" by the fusion routines.

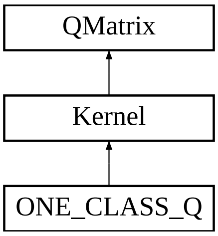
Definition at line 297 of file anomaly_detection.h.

The documentation for this struct was generated from the following file:

- anomaly_detection.h

4.15 Kernel Class Reference

Inheritance diagram for Kernel:



Public Member Functions

- **Kernel** (int l, svm_node ∗const ∗x, const svm_parameter &param)
- virtual Qfloat ∗ **get_Q** (int column, int len) const =0
- virtual double ∗ **get_QD** () const =0
- virtual void **swap_index** (int i, int j) const

Static Public Member Functions

- static double **k_function** (const svm_node ∗x, const svm_node ∗y, const svm_parameter &param)

Protected Attributes

- double(Kernel::∗ **kernel_function**)(int i, int j) const

4.15.1 Detailed Description

Definition at line 188 of file svm.cpp.

The documentation for this class was generated from the following file:

- svm.cpp

4.16 MagBuffer Struct Reference

```
#include <magnetic.h>
```

Data Fields

- int16_t iBs [3][MAGBUFFSIZEX][MAGBUFFSIZEY]
  uncalibrated magnetometer readings
- int32_t index [MAGBUFFSIZEX][MAGBUFFSIZEY]
  array of time indices
- int16_t tanarray [MAGBUFFSIZEX - 1]
  array of tangents of (100 ∗ angle)
- int16_t iMagBufferCount
  number of magnetometer readings

4.16.1 Detailed Description

The Magnetometer Measurement Buffer holds a 3-dimensional "constellation" of data points.

The constellation of points are used to compute magnetic hard/soft iron compensation terms. The contents of this buffer are updated on a continuing basis.

Definition at line 68 of file magnetic.h.

The documentation for this struct was generated from the following file:

- magnetic.h

4.17 MagCalibration Struct Reference

Magnetic Calibration Structure.

```
#include <magnetic.h>
```

Data Fields

- float fV [3]

  current hard iron offset x, y, z, (uT)
- float finvW [3][3]

  current inverse soft iron matrix
- float fB

  current geomagnetic field magnitude (uT)
- float fBSq

  square of fB (uT^2)
- float fFitErrorpc

  current fit error %
- int32_t iValidMagCal

  solver used: 0 (no calibration) or 4, 7, 10 element
- float ftrV [3]

  trial value of hard iron offset z, y, z (uT)
- float ftrinvW [3][3]

  trial inverse soft iron matrix size
- float ftrB

  trial value of geomagnetic field magnitude in uT
- float ftrFitErrorpc

  trial value of fit error %
- float fA [3][3]

  ellipsoid matrix A
- float finvA [3][3]

  inverse of ellipsoid matrix A
- float fmatA [10][10]

  scratch 10x10 float matrix used by calibration algorithms
- float fmatB [10][10]

  scratch 10x10 float matrix used by calibration algorithms

- float fvecA [10]

  scratch 10x1 vector used by calibration algorithms
- float fvecB [4]

  scratch 4x1 vector used by calibration algorithms
- float fYTY

  Y^T.Y for 4 element calibration = (iB^2)^2.
- int32_t iSumBs [3]

  sum of measurements in buffer (counts)
- int32_t iMeanBs [3]

  average magnetic measurement (counts)
- int32_t itimeslice

  counter for tine slicing magnetic calibration calculations
- int8_t iCalInProgress

  flag denoting that a calibration is in progress
- int8_t iNewCalibrationAvailable

  flag denoting that a new calibration has been computed
- int8_t iInitiateMagCal

  flag to start a new magnetic calibration
- int8_t iMagBufferReadOnly

  flag to denote that the magnetic measurement buffer is temporarily read only
- int8_t i4ElementSolverTried

  flag to denote at least one attempt made with 4 element calibration
- int8_t i7ElementSolverTried

  flag to denote at least one attempt made with 4 element calibration
- int8_t i10ElementSolverTried

  flag to denote at least one attempt made with 4 element calibration

4.17.1 Detailed Description

Magnetic Calibration Structure.

Definition at line 77 of file magnetic.h.

The documentation for this struct was generated from the following file:

- magnetic.h

4.18 MagSensor Struct Reference

The MagSensor structure stores raw and processed measurements for a 3-axis magnetic sensor.

```
#include <anomaly_detection.h>
```

Data Fields

- uint8_t iWhoAmI

    sensor whoami
- bool isEnabled

    true if the device is sampling
- uint8_t iFIFOCount

    number of measurements read from FIFO
- uint16_t iFIFOExceeded

    Number of samples received in excess of software FIFO size.
- float fFloatPerCount

    uT per count
- float fSum [4]

    sum of all measurements in this epoch
- float fSum2 [4]

    sum of squares over all measurements in this epoch
- float features [NUM_FEATURES][4]

    rows are mean, variance, SF, kurtosis and mean crossing rate
- float fFIFO [MAG_FIFO_SIZE][4]

    FIFO measurements.

4.18.1 Detailed Description

The MagSensor structure stores raw and processed measurements for a 3-axis magnetic sensor.

The MagSensor structure stores raw and processed measurements, as well as metadata for a single 3-axis magnetometer. This structure is normally "fed" by the sensor driver and "consumed" by the fusion routines.

Definition at line 278 of file anomaly_detection.h.

The documentation for this struct was generated from the following file:

- anomaly_detection.h

4.19 Model Union Reference

A Union type model structure.

```
#include <machineLearning_subsystem.h>
```

Data Fields

- struct GmmModel **gmm_model**
- struct OcsvmModel **ocsvm_model**

4.19.1 Detailed Description

A Union type model structure.

Each modelInstance has one of these two types of models.

Definition at line 135 of file machineLearning_subsystem.h.

The documentation for this union was generated from the following file:

- machineLearning_subsystem.h

4.20 ModelCollection Struct Reference

Model Collection structure contains the poinsters of ModelInstance structures.

```
#include <machineLearning_subsystem.h>
```

Data Fields

- struct ModelInstance ∗ **pModelInstance** [MAX_MODEL_INSTANCES]
- uint16_t training_rate

  variable to define how often the model instance is trained. every "training_rate" feature samples.
- uint8_t model_IDs_ensemble [MAX_MODEL_INSTANCES]

  array that contains lower level model IDs.
- uint8_t ensemble_modelID

  ensemble model ID (range: 0x11 - 0x1F)
- uint8_t num_LL_models

  of lower level models
- uint8_t votes_required

  condition to fuse lower level model decisions.

4.20.1 Detailed Description

Model Collection structure contains the poinsters of ModelInstance structures.

Potentially, control parameters for a collection of Model Instances may be added more.

Definition at line 167 of file machineLearning_subsystem.h.

The documentation for this struct was generated from the following file:

- machineLearning_subsystem.h

4.21 ModelInstance Struct Reference

Model Instance Structure contains the buffers for feature samples and other information.

```
#include <machineLearning_subsystem.h>
```

Data Fields

- model_t model_type

  Machine learning model type defined in model_t.
- uint8_t modelID

  Model name (index) determined by a user in GUI.
- uint8_t iStage

  Stages for each algorithm. They are set by gbls->controlCommandforML.
- uint8_t feature_dimension

  Dimensionality of feature inputs for this model instance.
- uint8_t featureBufferSize

  This determins a window size, moving in fBufferNormalized. We may have different buffer use for different models.
- int startPositionBuffer

  Start position of a moving window.
- int endPositionBuffer

  End position of a moving window.
- bool reversedPosition

  An indicator whether the start-end positions are reversed. Note the buffer size limited, and the window moves in a circular type buffer.
- bool isEnabled

  True if the model instance is enabled.
- float threshold

  A threshold value to detect anomality.
- bool isTrained

  True once the model instance is trained.
- bool decision_hard

  Binary decision of anomaly.
- float decision_soft

  Soft decision such as output of a probability density function.
- bool **enableTransmit**
- struct FeatureInstance ∗ pFeature [MAX_FEATURE_DIMENSION]

  pointers that indicates feature instances
- union Model ∗ pModel

  A pointor that indicates a union of GMM and OC-SVM.

4.21.1 Detailed Description

Model Instance Structure contains the buffers for feature samples and other information.

Model type and ID are used to identify model instance. The selected feature instances are linked to a model instance. GMM and OC-SVM are linked to a model instance as a union. So, only one model is used in a model instance.

Definition at line 145 of file machineLearning_subsystem.h.

The documentation for this struct was generated from the following file:

- machineLearning_subsystem.h

4.22 OcsvmModel Struct Reference

A One Class Support Vector Machine (OC-SVM) structure.

```
#include <machineLearning_subsystem.h>
```

Data Fields

- uint8_t modelID

  The model number, assigned from GUI.
- uint8_t opt_num_SVs

  The optimal number of SVs decided after training.
- uint8_t KernelType

  A Kernel type is chosen from this list: LINEAR, RBF, POLY, SIGMOID, PRECOMPUTED. But current version only considers RBF.
- float KernelSize

  The kernel size sigma for RBF. In LIBSVM, RBF is defined as exp(- gamma ∗ x^2). Thus, gamma = 1 / 2 ∗ sigma^2.
- float nu

  Nu parameter for OC-SVM affects the bounds on the number SVs and fraction of anomaly. More details are in [Scholkopf et al., "Estimating the Support of a High-Dimensional Distriution].
- struct svm_node_embedded SV [MAX_SVM_SVs][MAX_FEATURE_DIMENSION]

  Collection of SVs.
- float sv_coef [MAX_SVM_SVs]

  Coefficients that correspond to the trained SVs.
- float rho

  Threshold computed after training.

4.22.1 Detailed Description

A One Class Support Vector Machine (OC-SVM) structure.

OC-SVM parameters as well as trained model by use of LIBSVM. This structure is linked to ModelInstance.

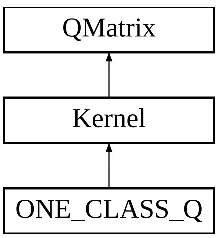
Definition at line 121 of file machineLearning_subsystem.h.

The documentation for this struct was generated from the following file:

- machineLearning_subsystem.h

4.23 ONE_CLASS_Q Class Reference

Inheritance diagram for ONE_CLASS_Q:



Public Member Functions

- **ONE_CLASS_Q** (const svm_problem &prob, const svm_parameter &param)
- Qfloat ∗ **get_Q** (int i, int len) const
- double ∗ **get_QD** () const
- void **swap_index** (int i, int j) const

Additional Inherited Members

4.23.1 Detailed Description

Definition at line 985 of file svm.cpp.

The documentation for this class was generated from the following file:

- svm.cpp

4.24 PhysicalSensor Struct Reference

An instance of PhysicalSensor structure type should be allocated for each physical sensors (combo devices = 1)

```
#include <anomaly_detection.h>
```

Data Fields

- registerDeviceInfo_t deviceInfo

    I2C device context.
- void ∗ bus_driver

    should be of type (ARM_DRIVER_I2C∗ for I2C-based sensors, ARM_DRIVER_SPI∗ for SPI)
- registerDeviceInfo_t ∗ busInfo

    information required for bus power management
- uint16_t addr

    I2C address if applicable.
- uint16_t isInitialized

    Bitfields to indicate sensor is active (use SensorBitFields from build.h)
- spiSlaveSpecificParams_t slaveParams

    SPI specific parameters. Not used for I2C.
- struct PhysicalSensor ∗ next

    pointer to next sensor in this linked list
- uint16_t schedule

    Parameter to control sensor sampling rate.
- initializeSensor_t ∗ initialize

    pointer to function to initialize sensor using the supplied drivers
- readSensor_t ∗ read

    pointer to function to read sensor using the supplied drivers

4.24.1 Detailed Description

An instance of PhysicalSensor structure type should be allocated for each physical sensors (combo devices = 1)

These structures sit 'on-top-of' the pre-7.0 sensor fusion structures and give us the ability to do run time driver installation.

Definition at line 221 of file anomaly_detection.h.

The documentation for this struct was generated from the following file:

- anomaly_detection.h

4.25 PressureSensor Struct Reference

The PressureSensor structure stores raw and processed measurements for an altimeter.

```
#include <anomaly_detection.h>
```

Data Fields

- uint8_t iWhoAmI
  sensor whoami
- bool isEnabled
  true if the device is sampling
- int32_t iH
  most recent unaveraged height (counts)
- int32_t iP
  most recent unaveraged pressure (counts)
- float fH
  most recent unaveraged height (m)
- float fT
  most recent unaveraged temperature (C)
- float fmPerCount
  meters per count
- float fCPerCount
  degrees Celsius per count
- int16_t iT
  most recent unaveraged temperature (counts)

4.25.1 Detailed Description

The PressureSensor structure stores raw and processed measurements for an altimeter.

The PressureSensor structure stores raw and processed measurements, as well as metadata for a pressure sensor/altimeter.

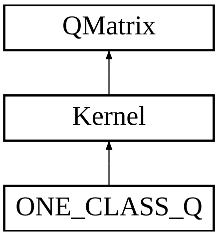
Definition at line 240 of file anomaly_detection.h.

The documentation for this struct was generated from the following file:

- anomaly_detection.h

4.26 QMatrix Class Reference

Inheritance diagram for QMatrix:



Public Member Functions

- virtual Qfloat ∗ **get_Q** (int column, int len) const =0
- virtual double ∗ **get_QD** () const =0
- virtual void **swap_index** (int i, int j) const =0

4.26.1 Detailed Description

Definition at line 180 of file svm.cpp.

The documentation for this class was generated from the following file:

- svm.cpp

4.27 Quaternion Struct Reference

quaternion structure definition

```
#include <orientation.h>
```

Data Fields

- float q0
  scalar component
- float q1
  x vector component
- float q2
  y vector component
- float q3
  z vector component

4.27.1 Detailed Description

quaternion structure definition

Definition at line 42 of file orientation.h.

The documentation for this struct was generated from the following file:

- orientation.h

4.28 Solver::SolutionInfo Struct Reference

Data Fields

- double **obj**
- double **rho**
- double **upper_bound_p**
- double **upper_bound_n**
- double **r**

4.28.1 Detailed Description

Definition at line 384 of file svm.cpp.

The documentation for this struct was generated from the following file:

- svm.cpp

4.29 Solver Class Reference

Data Structures

- struct SolutionInfo

Public Member Functions

- void **Solve** (int l, const QMatrix &Q, const double ∗p_, const schar ∗y_, double ∗alpha_, double Cp, double Cn, double eps, SolutionInfo ∗si, int shrinking)

Protected Types

- enum { **LOWER_BOUND**, **UPPER_BOUND**, **FREE** }

Protected Member Functions

- double **get_C** (int i)
- void **update_alpha_status** (int i)
- bool **is_upper_bound** (int i)
- bool **is_lower_bound** (int i)
- bool **is_free** (int i)
- void **swap_index** (int i, int j)
- void **reconstruct_gradient** ()
- virtual int **select_working_set** (int &i, int &j)
- virtual double **calculate_rho** ()
- virtual void **do_shrinking** ()

Protected Attributes

- int **active_size**
- schar ∗ **y**
- double ∗ **G**
- char ∗ **alpha_status**
- double ∗ **alpha**
- const QMatrix ∗ **Q**
- const double ∗ **QD**
- double **eps**
- double **Cp**
- double **Cn**
- double ∗ **p**
- int ∗ **active_set**
- double ∗ **G_bar**
- int **l**
- bool **unshrink**

4.29.1 Detailed Description

Definition at line 379 of file svm.cpp.

The documentation for this class was generated from the following file:

- svm.cpp

4.30 StatusSubsystem Struct Reference

StatusSubsystem() provides an object-like interface for communicating status to the user.

```
#include <status.h>
```

Data Fields

- ad_status_t previous

    Previous status state - ad_status_t is defined in anomaly_detection.h.
- ad_status_t status

    Current status.
- ad_status_t next

    Pending status change.
- ssSetStatus_t * set

    change status immediately - no delay
- ssSetStatus_t * queue

    queue status change for next regular interval
- ssUpdateStatus_t * update

    make pending status active/visible
- ssUpdateStatus_t * test

    unit test which simply increments to next state
- uint8_t toggle

    This implementation can change LED color and have either solid/toggle.

4.30.1 Detailed Description

StatusSubsystem() provides an object-like interface for communicating status to the user.

Definition at line 44 of file status.h.

The documentation for this struct was generated from the following file:

- status.h

4.31 svm_model Struct Reference

smv_model structure stores the model obtained from the training procedure.

```
#include <svm.h>
```

Data Fields

- struct svm_parameter **param**
- int **nr_class**
- int **l**
- struct svm_node ** **SV**
- double ** **sv_coef**
- double * **rho**
- double * **probA**
- double * **probB**
- int * **sv_indices**
- int * **label**
- int * **nSV**
- int **free_sv**

4.31.1 Detailed Description

smv_model structure stores the model obtained from the training procedure.

Definition at line 74 of file svm.h.

The documentation for this struct was generated from the following file:

- svm.h

4.32 svm_node Struct Reference

Each svm_node represents an element of multidimensional features.

```
#include <svm.h>
```

Data Fields

- int **index**
- double **value**

4.32.1 Detailed Description

Each svm_node represents an element of multidimensional features.

Index starts from 1 upto the dimensionality of feature sample. If index is set to -1, then the svm_node is the end of the SV. double value contains the value of SV.

Definition at line 26 of file svm.h.

The documentation for this struct was generated from the following file:

- svm.h

4.33 svm_node_embedded Struct Reference

A structure for each support vector.

```
#include <machineLearning_subsystem.h>
```

Data Fields

- uint8_t index

  Index indicates the dimensionality.
- float value

  A single dimensional value of a SV.

4.33.1 Detailed Description

A structure for each support vector.

This structure was defined for embedded systems with reduced memory usage, rather than using 8 bytes double type. It follows the same structure of LIBSVM.

Definition at line 112 of file machineLearning_subsystem.h.

The documentation for this struct was generated from the following file:

- machineLearning_subsystem.h

4.34 svm_parameter Struct Reference

svm_parameter contains the parameters for a SVM.

```
#include <svm.h>
```

Data Fields

- int **svm_type**
- int **kernel_type**
- int **degree**
- double **gamma**
- double **coef0**
- double **cache_size**
- double **eps**
- double **C**
- int **nr_weight**
- int * **weight_label**
- double * **weight**
- double **nu**
- double **p**
- int **shrinking**
- int **probability**

4.34.1 Detailed Description

svm_parameter contains the parameters for a SVM.

OC-SVM parameters are put by parse_svm_param().

Definition at line 51 of file svm.h.

The documentation for this struct was generated from the following file:

- svm.h

4.35 svm_problem Struct Reference

svm_problem describes a problem with a SVM.

```
#include <svm.h>
```

Data Fields

- int **l**
- double $*$ **y**
- struct svm_node $**$ **x**

4.35.1 Detailed Description

svm_problem describes a problem with a SVM.

l denotes the number of training data. y denotes an array that contains the target information. For OC-SVM, only the single label 1 is considered. x denotes an array of pointers. Each pointer indicates a training data vector.

Definition at line 37 of file svm.h.

The documentation for this struct was generated from the following file:

- svm.h

Chapter 5

File Documentation

5.1 anomaly_detection.c File Reference

The anomaly_detection.c file implements the top level programming interface.

```
#include <stdio.h>
#include <math.h>
#include "anomaly_detection.h"
#include "magnetic.h"
#include "drivers.h"
#include "sensor_drv.h"
#include "status.h"
#include "control.h"
#include "fusion.h"
#include "fsl_debug_console.h"
#include "timers.h"
```

Functions

- void setStatus (Globals ∗gbls, ad_status_t status)
- void queueStatus (Globals ∗gbls, ad_status_t status)
- void updateStatus (Globals ∗gbls)
- void **testStatus** (Globals ∗gbls)
- void initGlobals (Globals ∗gbls, StatusSubsystem ∗pStatusSubsystem, ControlSubsystem ∗pControlSubsystem, FeatureCollection ∗pFeatureCollection, ModelCollection ∗pModelCollection)

  utility function to insert default values in the top level structure
- int8_t installSensor (Globals ∗gbls, struct PhysicalSensor ∗pSensor, uint16_t addr, uint16_t schedule, void ∗bus_driver, registerDeviceInfo_t ∗busInfo, initializeSensor_t ∗initialize, readSensor_t ∗read)
- int8_t **initializeSensors** (Globals ∗gbls)
- void **computeMagFeatures** (Globals ∗gbls)
- int8_t readSensors (Globals ∗gbls, uint16_t read_loop_counter)
- void computeFeatures (Globals ∗gbls)
- void **process_ctr_command** (Globals ∗gbls)
- void **process_ML_command** (Globals ∗gbls)
- void zeroArray (StatusSubsystem ∗pStatus, void ∗data, uint16_t size, uint16_t numElements, uint8_t check)
- void clearFIFOs (Globals ∗gbls)

  Function to clear FIFO at the end of each fusion computation.

- void runAD (Globals *gbls)
- void initializeAD (Globals *gbls)
- uint16 **sign** (float x)
- float **max** (float a, float b)
- void computeBasicFeatures (union FifoSensor *sensor)
- void addToFifo (union FifoSensor *sensor, uint16_t maxFifoSize, int16_t sample[3])

  addToFifo is called from within sensor driver read functions
- void **clearFeatureBuffers** (struct ModelCollection *modelCollection)

Variables

- float **training_time_used**
- uint32_t **xTimeNow**
- uint32_t **xTimeElapsed**
- uint16_t feature_interval_number = 0

  ApplyPerturbation is a reverse unit-step test function.

5.1.1 Detailed Description

The anomaly_detection.c file implements the top level programming interface.

5.1.2 Function Documentation

5.1.2.1 addToFifo()

```
void addToFifo (
            union FifoSensor * sensor,
            uint16_t maxFifoSize,
            int16_t sample[3] )
```

addToFifo is called from within sensor driver read functions

addToFifo is called from within sensor driver read functions to transfer new readings into the sensor structure corresponding to accel, gyro or mag. This function ensures that the software FIFOs are not overrun.

example usage: if (status==SENSOR_ERROR_NONE) addToFifo((FifoSensor*) &(gbls->Mag), MAG_FIFO_SIZE, sample);

Parameters

sensor	pointer to structure of type AccelSensor, MagSensor or GyroSensor
maxFifoSize	the size of the software (not hardware) FIFO
sample	the sample to add

Definition at line 499 of file anomaly_detection.c.

5.1.2.2 clearFIFOs()

```
void clearFIFOs (
            Globals * gbls )
```

Function to clear FIFO at the end of each fusion computation.

Parameters

gbls	Global data structure pointer

Definition at line 374 of file anomaly_detection.c.

5.1.2.3 computeBasicFeatures()

```
void computeBasicFeatures (
            union FifoSensor * sensor )
```

Parameters

sensor	logical sensor data structure pointer

Definition at line 450 of file anomaly_detection.c.

5.1.2.4 computeFeatures()

```
void computeFeatures (
            Globals * gbls )
```

conditionSensorReadings() transforms raw software FIFO readings into forms that can be consumed by the sensor fusion engine. This include sample averaging and (in the case of the gyro) integrations, applying hardware abstraction layers, and calibration functions. This function is normally involved via the "gbls." global pointer.

Parameters

gbls	Global data structure pointer

Definition at line 249 of file anomaly_detection.c.

5.1.2.5 initGlobals()

```
void initGlobals (
            Globals * gbls,
```

```
StatusSubsystem * pStatusSubsystem,
ControlSubsystem * pControlSubsystem,
FeatureCollection * pFeatureCollection,
ModelCollection * pModelCollection )
```

utility function to insert default values in the top level structure

Parameters

gbls	Global data structure pointer
pStatusSubsystem	Status subsystem pointer
pControlSubsystem	Control subsystem pointer
pFeatureCollection	Feature collection
pModelCollection	Model collection

Definition at line 81 of file anomaly_detection.c.

5.1.2.6 initializeAD()

```
void initializeAD (
        Globals * gbls )
```

This function is responsible for initializing the system prior to starting the main fusion loop. This function is normally involved via the "gbls." global pointer.

Definition at line 417 of file anomaly_detection.c.

5.1.2.7 installSensor()

```
int8_t installSensor (
        Globals * gbls,
        struct PhysicalSensor * pSensor,
        uint16_t addr,
        uint16_t schedule,
        void * bus_driver,
        registerDeviceInfo_t * busInfo,
        initializeSensor_t * initialize,
        readSensor_t * read )
```

installSensor is used to instantiate a physical sensor driver into the sensor fusion system. This function is normally involved via the "gbls." global pointer.

Parameters

gbls	top level fusion structure
pSensor	pointer to structure describing physical sensor
addr	I2C address for sensor (if applicable)
schedule	Parameter to control sensor sampling rate
bus_driver	ISSDK sensor bus driver (usually KSDK I2C bus)
busInfo	information required for bus power management
initialize	pointer to sensor initialization function
read	pointer to sensor read function

Definition at line 130 of file anomaly_detection.c.

5.1.2.8 queueStatus()

```
void queueStatus (
            Globals * gbls,
            ad_status_t status )
```

Poor man's inheritance for status subsystem queueStatus command. This function is normally involved via the "gbls." global pointer.

Definition at line 63 of file anomaly_detection.c.

5.1.2.9 readSensors()

```
int8_t readSensors (
            Globals * gbls,
            uint16_t read_loop_counter )
```

readSensors traverses the linked list of physical sensors, calling the individual read functions one by one. This function is normally involved via the "gbls." global pointer.

Parameters

gbls	pointer to global sensor fusion data structure
read_loop_counter	current loop counter (used for multirate processing)

Definition at line 220 of file anomaly_detection.c.

5.1.2.10 runAD()

```
void runAD (
            Globals * gbls )
```

runAD the top level call that actually runs the sensor fusion. This is a utility function which manages the various defines in build.h. You should feel free to drop down a level and implement only those portions of fFuseSensors() that your application needs. This function is normally involved via the "gbls." global pointer.

Definition at line 408 of file anomaly_detection.c.

5.1.2.11 setStatus()

```
void setStatus (
            Globals * gbls,
            ad_status_t status )
```

Poor man's inheritance for status subsystem setStatus command This function is normally involved via the "gbls." global pointer.

Definition at line 56 of file anomaly_detection.c.

5.1.2.12 updateStatus()

```
void updateStatus (
            Globals * gbls )
```

Poor man's inheritance for status subsystem updateStatus command. This function is normally involved via the "gbls." global pointer.

Definition at line 70 of file anomaly_detection.c.

5.1.2.13 zeroArray()

```
void zeroArray (
            StatusSubsystem * pStatus,
            void * data,
            uint16_t size,
            uint16_t numElements,
            uint8_t check )
```

Parameters

pStatus	Status subsystem pointer
data	pointer to array to be zeroed
size	data type size = 8, 16 or 32
numElements	number of elements to zero out
check	true if you would like to verify writes, false otherwise

Definition at line 331 of file anomaly_detection.c.

5.1.3 Variable Documentation

5.1.3.1 feature_interval_number

```
uint16_t feature_interval_number = 0
```

ApplyPerturbation is a reverse unit-step test function.

The ApplyPerturbation function applies a user-specified step function to prior fusion results which is then "released" in the next fusion cycle. When used in conjuction with the NXP Sensor Fusion Toolbox, this provides a visual indication of the dynamic behavior of the library. ApplyPerturbation() is defined in debug.c.

Definition at line 52 of file anomaly_detection.c.

5.2 anomaly_detection.h File Reference

The anomaly_detection.h file implements the top level programming interface.

```
#include "math.h"
#include "stdbool.h"
#include "stdio.h"
#include "stdint.h"
#include "issdk_hal.h"
#include "build.h"
#include "orientation.h"
#include "register_io_spi.h"
#include "matrix.h"
#include "machineLearning_subsystem.h"
#include "output_stream.h"
```

Data Structures

- struct PhysicalSensor

  An instance of PhysicalSensor structure type should be allocated for each physical sensors (combo devices = 1)
- struct PressureSensor

  The PressureSensor structure stores raw and processed measurements for an altimeter.
- struct AccelSensor

  The AccelSensor structure stores raw and processed measurements for a 3-axis accelerometer.
- struct MagSensor

  The MagSensor structure stores raw and processed measurements for a 3-axis magnetic sensor.
- struct GyroSensor

  The GyroSensor structure stores raw and processed measurements for a 3-axis gyroscope.
- union FifoSensor

  The FifoSensor union allows us to use common pointers for Accel, Mag & Gyro logical sensor structures.
- struct Globals

  The top level fusion structure.

Macros

- #define true 1

  Boolean TRUE.
- #define false 0

  Boolean FALSE.
- #define CI_ACCELEROMETER_X 0x01

  Application-specific serial communications system.
- #define **CI_ACCELEROMETER_Y** 0x02
- #define **CI_ACCELEROMETER_Z** 0x03
- #define **CI_ACCELEROMETER_VM** 0x04
- #define **CI_GYRO_X** 0x05
- #define **CI_GYRO_Y** 0x06
- #define **CI_GYRO_Z** 0x07
- #define **CI_GYRO_VM** 0x08
- #define **CI_TEMPERATURE** 0x09
- #define **CI_PRESSURE** 0x0A
- #define **CI_MICROPHONE** 0x0B
- #define **CI_MEAN** 0x0040
- #define **CI_VARIANCE** 0x0020
- #define **CI_SKEW_FACTOR** 0x0010
- #define **CI_KURTOSIS** 0x0008
- #define **CI_CROSSING_RATE** 0x0004
- #define **CI_STD** 0x0002
- #define **CI_NORMALIZED_STD** 0x0001

Generic bit-field values

Generic bit-field values

- #define **B0** (1 << 0)
- #define **B1** (1 << 1)
- #define **B2** (1 << 2)
- #define **B3** (1 << 3)

Math Constants

useful multiplicative conversion constants

- #define PI 3.141592654F

  pi
- #define PIOVER2 1.570796327F

  pi / 2
- #define FPIOVER180 0.01745329251994F

  degrees to radians conversion = pi / 180
- #define F180OVERPI 57.2957795130823F

  radians to degrees conversion = 180 / pi
- #define F180OVERPISQ 3282.8063500117F

  square of F180OVERPI
- #define ONETHIRD 0.33333333F

  one third
- #define ONESIXTH 0.166666667F

  one sixth
- #define ONESIXTEENTH 0.0625F

  one sixteenth
- #define ONEOVER12 0.083333333F

  1 / 12

- #define ONEOVER48 0.02083333333F

  1 / 48
- #define ONEOVER120 0.0083333333F

  1 / 120
- #define ONEOVER3840 0.0002604166667F

  1 / 3840
- #define ONEOVERSQRT2 0.707106781F

  1/sqrt(2)
- #define SQRT15OVER4 0.968245837F

  sqrt(15)/4
- #define GTOMSEC2 9.80665

  standard gravity in m/s2

Typedefs

Integer Typedefs

Typedefs to map common integer types to standard form

- typedef unsigned char **byte**
- typedef int8_t **int8**
- typedef int16_t **int16**
- typedef int32_t **int32**
- typedef uint8_t **uint8**
- typedef uint16_t **uint16**
- typedef uint32_t **uint32**

Enumerations

- enum model_t { **NO_MODEL** =0, GMM, OCSVM, **ENSEMBLE** }

  Machine Learning Algorithms types.

Vector Components

Index values for accessing vector terms

- #define **NUM_FEATURES** (END_OF_FEATURES-1)
- #define **SPI_ADDR** 0x00
- enum **axis_t** {
  CHX = 0, **CHY**, **CHZ**, **VM**,
  SCALAR, **END_OF_AXIS_TYPES** }
- enum **sensor_t** {
  ACCEL = 0, **GYRO**, **MAG**, **TEMPERATURE**,
  PRESSURE, **MICROPHONE**, **END_OF_SENSORS** }
- enum **feature_t** {
  MEAN = 0, **VARIANCE**, **SKEW_FACTOR**, **KURTOSIS**,
  CROSSING_RATE, **STD**, **NORMALIZED_STD**, **CREST_FACTOR**,
  END_OF_FEATURES }
- enum ad_status_t {
  OFF, INITIALIZING, LOWPOWER, NORMAL,
  RECEIVING, SENDING, HARD_FAULT, SOFT_FAULT }
- enum **command_t** {
  NO_COMMAND = 0x00, **CLR_COMMAND** = 0x01, **STOP_COMMAND** = 0x02, **RUN_COMMAND** = 0x04,
  TRAIN_COMMAND = 0x08, **DELETE_COMMAND** = 0x10, **DELETE_ALL_COMMAND** = 0x20, **TED_CO↩**
  MMAND = 0x40 }

- typedef int8_t() **initializeSensor_t**(struct PhysicalSensor ∗sensor, struct Globals ∗gbls)
- typedef int8_t() **readSensor_t**(struct PhysicalSensor ∗sensor, struct Globals ∗gbls)
- typedef int8_t() **readSensors_t**(struct Globals ∗gbls, uint16_t read_loop_counter)
- typedef int8_t() **installSensor_t**(struct Globals ∗gbls, struct PhysicalSensor ∗sensor, uint16_t addr, uint16←_t schedule, void ∗bus_driver, registerDeviceInfo_t ∗busInfo, initializeSensor_t ∗initialize, readSensor_←t ∗read)
- typedef void() **initializeFusionEngine_t**(struct Globals ∗gbls)
- typedef void() **runFusion_t**(struct Globals ∗gbls)
- typedef void() **clearFIFOs_t**(struct Globals ∗gbls)
- typedef void() **setStatus_t**(struct Globals ∗gbls, ad_status_t status)
- typedef void() **updateStatus_t**(struct Globals ∗gbls)
- typedef void() **ssSetStatus_t**(struct StatusSubsystem ∗pStatus, ad_status_t status)
- typedef void() **ssUpdateStatus_t**(struct StatusSubsystem ∗pStatus)
- typedef struct Globals Globals

    The top level fusion structure.
- installSensor_t **installSensor**
- initializeFusionEngine_t **initializeAD**
- runFusion_t **runAD**
- readSensors_t **readSensors**
- uint16_t feature_interval_number

    ApplyPerturbation is a reverse unit-step test function.
- void initGlobals (Globals ∗gbls, struct StatusSubsystem ∗pStatusSubsystem, struct ControlSubsystem ∗p←ControlSubsystem, struct FeatureCollection ∗pFeatureCollection, struct ModelCollection ∗pModelCollection)

    utility function to insert default values in the top level structure
- void computeFeatures (Globals ∗gbls)
- void computeBasicFeatures (union FifoSensor ∗sensor)
- void clearFIFOs (Globals ∗gbls)

    Function to clear FIFO at the end of each fusion computation.
- void zeroArray (struct StatusSubsystem ∗pStatus, void ∗data, uint16_t size, uint16_t numElements, uint8_t check)
- void addToFifo (union FifoSensor ∗sensor, uint16_t maxFifoSize, int16_t sample[3])

    addToFifo is called from within sensor driver read functions
- void ApplyAccelHAL (struct AccelSensor ∗Accel)

    Apply the accelerometer Hardware Abstraction Layer.
- void ApplyMagHAL (struct MagSensor ∗Mag)

    Apply the magnetometer Hardware Abstraction Layer.
- void ApplyGyroHAL (struct GyroSensor ∗Gyro)

    Apply the gyroscope Hardware Abstraction Layer.
- void **process_ctr_command** (Globals ∗gbls)
- void **process_ML_command** (Globals ∗gbls)
- void **clearFeatureBuffers** (struct ModelCollection ∗modelCollection)

5.2.1 Detailed Description

The anomaly_detection.h file implements the top level programming interface.

5.2.2 Typedef Documentation

5.2.2.1 Globals

```
typedef struct Globals Globals
```

The top level fusion structure.

The top level fusion structure grows/shrinks based upon flag definitions contained in build.h. These same flags will populate the .iFlags field for run-time access.

5.2.3 Enumeration Type Documentation

5.2.3.1 ad_status_t

```
enum ad_status_t
```

Enumerator

OFF	These are the state definitions for the status subsystem. Application hasn't started
INITIALIZING	Initializing sensors and algorithms.
LOWPOWER	Running in reduced power mode.
NORMAL	Operation is Nominal.
RECEIVING	Receiving commands over wired interface (momentary)
SENDING	Sending data over wireless interface (momentary)
HARD_FAULT	Non-recoverable FAULT = something went very wrong.
SOFT_FAULT	Recoverable FAULT = something went wrong, but we can keep going.

Definition at line 161 of file anomaly_detection.h.

5.2.3.2 model_t

```
enum model_t
```

Machine Learning Algorithms types.

Enumerator

GMM	gaussian mixture model. EM algorithm will be used.
OCSVM	one-class SVM is assumed.

Definition at line 128 of file anomaly_detection.h.

5.2.4 Function Documentation

5.2.4.1 addToFifo()

```
void addToFifo (
            union FifoSensor * sensor,
            uint16_t maxFifoSize,
            int16_t sample[3] )
```

addToFifo is called from within sensor driver read functions

addToFifo is called from within sensor driver read functions to transfer new readings into the sensor structure corresponding to accel, gyro or mag. This function ensures that the software FIFOs are not overrun.

example usage: if (status==SENSOR_ERROR_NONE) addToFifo((FifoSensor∗) &(gbls->Mag), MAG_FIFO_SIZE, sample);

Parameters

sensor	pointer to structure of type AccelSensor, MagSensor or GyroSensor
maxFifoSize	the size of the software (not hardware) FIFO
sample	the sample to add

Definition at line 499 of file anomaly_detection.c.

5.2.4.2 ApplyAccelHAL()

```
void ApplyAccelHAL (
            struct AccelSensor * Accel )
```

Apply the accelerometer Hardware Abstraction Layer.

Parameters

Accel	pointer to accelerometer logical sensor

Definition at line 44 of file hal_frdm_fxs_mult2_b.c.

5.2.4.3 ApplyGyroHAL()

```
void ApplyGyroHAL (
            struct GyroSensor * Gyro )
```

Apply the gyroscope Hardware Abstraction Layer.

Parameters

Gyro	pointer to gyroscope logical sensor

Definition at line 99 of file hal_frdm_fxs_mult2_b.c.

5.2.4.4 ApplyMagHAL()

```
void ApplyMagHAL (
            struct MagSensor * Mag )
```

Apply the magnetometer Hardware Abstraction Layer.

Parameters

Mag	pointer to magnetometer logical sensor

Definition at line 71 of file hal_frdm_fxs_mult2_b.c.

5.2.4.5 clearFIFOs()

```
void clearFIFOs (
            Globals * gbls )
```

Function to clear FIFO at the end of each fusion computation.

Parameters

gbls	Global data structure pointer

Definition at line 374 of file anomaly_detection.c.

5.2.4.6 computeBasicFeatures()

```
void computeBasicFeatures (
            union FifoSensor * sensor )
```

Parameters

sensor	logical sensor data structure pointer

Definition at line 450 of file anomaly_detection.c.

5.2.4.7 computeFeatures()

```
void computeFeatures (
            Globals * gbls )
```

computeFeatures() transforms raw software FIFO readings into features to be used as input to the ML routines

conditionSensorReadings() transforms raw software FIFO readings into forms that can be consumed by the sensor fusion engine. This include sample averaging and (in the case of the gyro) integrations, applying hardware abstraction layers, and calibration functions. This function is normally involved via the "gbls." global pointer.

Parameters

gbls	Global data structure pointer

Definition at line 249 of file anomaly_detection.c.

5.2.4.8 initGlobals()

```
void initGlobals (
            Globals * gbls,
            struct StatusSubsystem * pStatusSubsystem,
            struct ControlSubsystem * pControlSubsystem,
            struct FeatureCollection * pFeatureCollection,
            struct ModelCollection * pModelCollection )
```

utility function to insert default values in the top level structure

Parameters

gbls	Global data structure pointer
pStatusSubsystem	Status subsystem pointer
pControlSubsystem	Control subsystem pointer
pFeatureCollection	Feature collection
pModelCollection	Model collection

Definition at line 81 of file anomaly_detection.c.

5.2.4.9 zeroArray()

```
void zeroArray (
            struct StatusSubsystem * pStatus,
```

```
void * data,
uint16_t size,
uint16_t numElements,
uint8_t check )
```

Parameters

pStatus	Status subsystem pointer
data	pointer to array to be zeroed
size	data type size = 8, 16 or 32
numElements	number of elements to zero out
check	true if you would like to verify writes, false otherwise

Definition at line 331 of file anomaly_detection.c.

5.2.5 Variable Documentation

5.2.5.1 feature_interval_number

```
uint16_t feature_interval_number
```

ApplyPerturbation is a reverse unit-step test function.

The ApplyPerturbation function applies a user-specified step function to prior fusion results which is then "released" in the next fusion cycle. When used in conjuction with the NXP Sensor Fusion Toolbox, this provides a visual indication of the dynamic behavior of the library. ApplyPerturbation() is defined in debug.c.

Definition at line 52 of file anomaly_detection.c.

5.3 approximations.c File Reference

Math approximations file.

```
#include "math.h"
#include "stdlib.h"
#include "stdint.h"
#include "approximations.h"
```

Macros

- #define **TAN15DEG** 0.26794919243F
- #define **TAN30DEG** 0.57735026919F
- #define **PADE_A** 96.644395816F
- #define **PADE_B** 25.086941612F
- #define **PADE_C** 1.6867633134F

Functions

- float **fasin_deg** (float x)
- float **facos_deg** (float x)
- float **fatan_deg** (float x)
- float **fatan2_deg** (float y, float x)
- float **fatan_15deg** (float x)

5.3.1 Detailed Description

Math approximations file.

Significant efficiencies were found by creating a set of trig functions which trade off precision for improved power/↩
CPU performance. Full details are included in Application Note AN5015: Trigonometry Approximations

5.4 approximations.h File Reference

Math approximations file.

Functions

- float **fasin_deg** (float x)
- float **facos_deg** (float x)
- float **fatan_deg** (float x)
- float **fatan2_deg** (float y, float x)
- float **fatan_15deg** (float x)

5.4.1 Detailed Description

Math approximations file.

Significant efficiencies were found by creating a set of trig functions which trade off precision for improved power/↩
CPU performance. Full details are included in Application Note AN5015: Trigonometry Approximations

5.5 board_encodings.h File Reference

This file summarizes board encodings assigned to date.

Macros

- #define **RESERVED0** 0
- #define **FRDM_KL25Z** 1
- #define **FRDM_K20D50M** 2
- #define **XPRESSO_LPC11U68** 3
- #define **FRDM_KL26Z** 4
- #define **FRDM_K64F** 5
- #define **XPRESSO_LPC1549** 6
- #define **FRDM_KL46Z** 7
- #define **FRDM_KW24D** 8
- #define **FRDM_K22F** 9
- #define **FRDM_KEAZ128** 10
- #define **XPRESSO_LPC4337** 11
- #define **FRDM_KV31F** 12
- #define **XPRESSO_LPC54102** 13
- #define **FRDM_KE02Z** 14
- #define **FRDM_KE06Z** 15
- #define **XPRESSO_LPC5411X** 16
- #define **FRDM_KL02Z** 17
- #define **FRDM_KL05Z** 18
- #define **FRDM_KW01Z** 19

5.5.1 Detailed Description

This file summarizes board encodings assigned to date.

It is not included within any of the sensor fusion code, and is provided for reference purposes only.

5.6 calibration_storage.c File Reference

Provides functions to store calibration to NVM.

```
#include <stdio.h>
#include "anomaly_detection.h"
#include "driver_KSDK_NVM.h"
#include "calibration_storage.h"
```

Functions

- void **SaveMagCalibrationToNVM** (Globals ∗gbls)
- void **SaveGyroCalibrationToNVM** (Globals ∗gbls)
- void **SaveAccelCalibrationToNVM** (Globals ∗gbls)
- void **EraseMagCalibrationFromNVM** (void)
- void **EraseGyroCalibrationFromNVM** (void)
- void **EraseAccelCalibrationFromNVM** (void)

5.6.1 Detailed Description

Provides functions to store calibration to NVM.

Users who are not using NXP hardware will need to supply their own drivers in place of those defined here.

5.7 calibration_storage.h File Reference

Provides functions to store calibration to NVM.

Functions

- void **SaveMagCalibrationToNVM** (Globals *gbls)
- void **SaveGyroCalibrationToNVM** (Globals *gbls)
- void **SaveAccelCalibrationToNVM** (Globals *gbls)
- void **EraseMagCalibrationFromNVM** (void)
- void **EraseGyroCalibrationFromNVM** (void)
- void **EraseAccelCalibrationFromNVM** (void)

5.7.1 Detailed Description

Provides functions to store calibration to NVM.

Users who are not using NXP hardware will need to supply their own drivers in place of those defined here.

5.8 control.c File Reference

Defines control sub-system.

```
#include "fsl_debug_console.h"
#include "board.h"
#include "pin_mux.h"
#include "fsl_uart.h"
#include "fsl_port.h"
#include "host_io_uart.h"
#include "anomaly_detection.h"
#include "host_interface_service.h"
#include "control.h"
```

Macros

- #define **UART_RX_RING_BUFFER_SIZE** 64

Functions

- void **clearCommands** (ControlSubsystem ∗pComm)
- int8_t initializeControlPort (ControlSubsystem ∗pComm)

  Initialize the control subsystem and all related hardware.

Variables

- uint8_t **gUartRxBuff**
- uint8_t **gHostRxBuff** [HOST_RX_BUF_LEN]
- volatile bool **bUartTxComplete**
- volatile bool **bUartRxPendingMsg**
- host_rx_packet_t **gHostRxPkt**
- host_channel_params_t **gHostChannelParams** [MAX_HOST_STREAMS]
- host_interface_handle_t **gHostHandle**
- Globals **gbls**
- uint8_t **gUartRingbuffer** [UART_RX_RING_BUFFER_SIZE] = {0}
- uart_handle_t **HOST_S_CMSIS_HANDLE**

5.8.1 Detailed Description

Defines control sub-system.

This file contains a UART implementation of the control subsystem. The command interpreter and streaming functions are contained in two separate files. So you can easily swap those out with only minor changes here.

5.8.2 Function Documentation

5.8.2.1 initializeControlPort()

```
int8_t initializeControlPort (
            ControlSubsystem * pComm )
```

Initialize the control subsystem and all related hardware.

Call this once to initialize structures, ports, etc. Initialize the UART driver.

Set UART Power mode.

Set UART Baud Rate.

Parameters

pComm	pointer to the control subystem structure

Definition at line 75 of file control.c.

5.9 control.h File Reference

Defines control sub-system.

Data Structures

- struct ControlSubsystem

  he ControlSubsystem encapsulates command and data streaming functions.

Typedefs

- typedef struct ControlSubsystem ControlSubsystem

  he ControlSubsystem encapsulates command and data streaming functions.

Functions

- int8_t initializeControlPort (ControlSubsystem ∗pComm)

  Call this once to initialize structures, ports, etc.

5.9.1 Detailed Description

Defines control sub-system.

Each sensor fusion application will probably have its own set of functions to control the fusion process and report results. This file defines the programming interface that should be followed in order for the fusion functions to operate correctly out of the box. The actual command interpreter is defined separately in DecodeCommandBytes.c. The output streaming function is defined in output_stream.c. Via these three files, the NXP Sensor Fusion Library provides a default set of functions which are compatible with the Sensor Fusion Toolbox. Use of the toolbox is highly recommended at least during initial development, as it provides many useful debug features. The NXP development team will typically require use of the toolbox as a pre-requisite for providing software support.

5.9.2 Typedef Documentation

5.9.2.1 ControlSubsystem

```
typedef struct ControlSubsystem ControlSubsystem
```

he ControlSubsystem encapsulates command and data streaming functions.

The ControlSubsystem encapsulates command and data streaming functions for the library. A C++-like typedef structure which includes executable methods for the subsystem is defined here.

5.9.3 Function Documentation

5.9.3.1 initializeControlPort()

```
int8_t initializeControlPort (
            ControlSubsystem * pComm )
```

Call this once to initialize structures, ports, etc.

Call this once to initialize structures, ports, etc. Initialize the UART driver.

Set UART Power mode.

Set UART Baud Rate.

Parameters

pComm	pointer to the control subystem structure

Definition at line 75 of file control.c.

5.10 control_lpsci.c File Reference

Defines control sub-system.

```
#include "fsl_debug_console.h"
#include "board.h"
#include "pin_mux.h"
#include "fsl_uart.h"
#include "fsl_lpsci.h"
#include "fsl_port.h"
#include "sensor_fusion.h"
#include "control.h"
```

Macros

- #define CONTROL_BAUDRATE 115200

  Baudrate to be used for serial communications.

Functions

- void **myUART_WriteByte** (UART0_Type ∗base, uint8_t data)
- int8_t **writeControlPort** (ControlSubsystem ∗pComm, uint8_t buffer[ ], uint16_t nbytes)
- void **BlueRadios_Init** (void)
- void **CONTROL_UART_IRQHandler** (void)
- int8_t initializeControlPort (ControlSubsystem ∗pComm)

  Initialize the control subsystem and all related hardware.

Variables

- uint8_t **sUARTOutputBuffer** [256]
- Globals **gbls**

5.10.1 Detailed Description

Defines control sub-system.

This file contains a Low power UART implementation of the control subsystem. This version is targeted specificially at FRDM-KL25Z, which utilizes a low power uart to drive both the OpenSDA and shield UART connections for FRDM-MULT2-B Bluetooth module.

The low power uart utilizes a slightly different interface within KSDK, hence this adaptation.

The command interpreter and streaming functions are contained in two separate files. So you can easily swap those out with only minor changes here.

5.10.2 Function Documentation

5.10.2.1 initializeControlPort()

```
int8_t initializeControlPort (
            ControlSubsystem * pComm )
```

Initialize the control subsystem and all related hardware.

Call this once to initialize structures, ports, etc. Initialize the UART driver.

Set UART Power mode.

Set UART Baud Rate.

Parameters

pComm	pointer to the control subystem structure

Definition at line 119 of file control_lpsci.c.

5.11 debug.c File Reference

ApplyPerturbation function used to analyze dynamic performance.

```
#include "anomaly_detection.h"
#include "control.h"
#include "stdlib.h"
#include "build.h"
```

Functions

- void ApplyPerturbation (Globals ∗gbls)

5.11.1 Detailed Description

ApplyPerturbation function used to analyze dynamic performance.

The ApplyPerturbation function applies a user-specified step function to prior fusion results which is then "released" in the next fusion cycle. When used in conjustion with the NXP Sensor Fusion Toolbox, this provides a visual indication of the dynamic behavior of the library.

Also included is some code for white-box testing within the IAR debug environment. It can be used to evaluate propagation delays for tilt and eCompass algorithms. It makes no sense with regard to "Rotation", because that algorithm is simple gyro integration, and will never return to the starting point. It will also overestimate delays for the kalman filters, as there is no actual gyro data corresponding to the simulated step function. So those filters are not operating as they would in the normal world.

5.11.2 Function Documentation

5.11.2.1 ApplyPerturbation()

```
void ApplyPerturbation (
            Globals * gbls )
```

The ApplyPerturbation function applies a user-specified step function to prior fusion results which is then "released" in the next fusion cycle. When used in conjustion with the NXP Sensor Fusion Toolbox, this provides a visual indication of the dynamic behavior of the library. This function is normally involved via the "gbls." global pointer.

Definition at line 58 of file debug.c.

5.12 debug.h File Reference

ApplyPerturbation function used to analyze dynamic performance.

Functions

- void ApplyPerturbation (Globals ∗gbls)

5.12.1 Detailed Description

ApplyPerturbation function used to analyze dynamic performance.

The ApplyPerturbation function applies a user-specified step function to prior fusion results which is then "released" in the next fusion cycle. When used in conjustion with the NXP Sensor Fusion Toolbox, this provides a visual indication of the dynamic behavior of the library.

5.12.2 Function Documentation

5.12.2.1 ApplyPerturbation()

```
void ApplyPerturbation (
            Globals * gbls )
```

The ApplyPerturbation function applies a user-specified step function to prior fusion results which is then "released" in the next fusion cycle. When used in conjustion with the NXP Sensor Fusion Toolbox, this provides a visual indication of the dynamic behavior of the library. This function is normally involved via the "gbls." global pointer.

Definition at line 58 of file debug.c.

5.13 DecodeCommandBytes.c File Reference

Command interpreter which interfaces to the Sensor Fusion Toolbox.

```
#include "anomaly_detection.h"
#include "control.h"
#include "fusion.h"
#include "calibration_storage.h"
```

Macros

- #define **cmd_VGplus** (((((('V' << 8) | 'G') << 8) | '+') << 8) | ' ')
- #define **cmd_VGminus** (((((('V' << 8) | 'G') << 8) | '-') << 8) | ' ')
- #define **cmd_DBplus** (((((('D' << 8) | 'B') << 8) | '+') << 8) | ' ')
- #define **cmd_DBminus** (((((('D' << 8) | 'B') << 8) | '-') << 8) | ' ')
- #define **cmd_Q3** (((((('Q' << 8) | '3') << 8) | ' ') << 8) | ' ')
- #define **cmd_Q3M** (((((('Q' << 8) | '3') << 8) | 'M') << 8) | ' ')
- #define **cmd_Q3G** (((((('Q' << 8) | '3') << 8) | 'G') << 8) | ' ')
- #define **cmd_Q6MA** (((((('Q' << 8) | '6') << 8) | 'M') << 8) | 'A')
- #define **cmd_Q6AG** (((((('Q' << 8) | '6') << 8) | 'A') << 8) | 'G')
- #define **cmd_Q9** (((((('Q' << 8) | '9') << 8) | ' ') << 8) | ' ')
- #define **cmd_RPCplus** (((((('R' << 8) | 'P') << 8) | 'C') << 8) | '+')
- #define **cmd_RPCminus** (((((('R' << 8) | 'P') << 8) | 'C') << 8) | '-')
- #define **cmd_ALTplus** (((((('A' << 8) | 'L') << 8) | 'T') << 8) | '+')
- #define **cmd_ALTminus** (((((('A' << 8) | 'L') << 8) | 'T') << 8) | '-')
- #define **cmd_RST** (((((('R' << 8) | 'S') << 8) | 'T') << 8) | ' ')
- #define **cmd_RINS** (((((('R' << 8) | 'I') << 8) | 'N') << 8) | 'S')
- #define **cmd_SVAC** (((((('S' << 8) | 'V') << 8) | 'A') << 8) | 'C')
- #define **cmd_SVMC** (((((('S' << 8) | 'V') << 8) | 'M') << 8) | 'C')
- #define **cmd_SVYC** (((((('S' << 8) | 'V') << 8) | 'Y') << 8) | 'C')
- #define **cmd_SVGC** (((((('S' << 8) | 'V') << 8) | 'G') << 8) | 'C')
- #define **cmd_ERAC** (((((('E' << 8) | 'R') << 8) | 'A') << 8) | 'C')
- #define **cmd_ERMC** (((((('E' << 8) | 'R') << 8) | 'M') << 8) | 'C')
- #define **cmd_ERYC** (((((('E' << 8) | 'R') << 8) | 'Y') << 8) | 'C')

- #define **cmd_ERGC** (((((('E' << 8) | 'R') << 8) | 'G') << 8) | 'C')
- #define **cmd_180X** (((((('1' << 8) | '8') << 8) | '0') << 8) | 'X')
- #define **cmd_180Y** (((((('1' << 8) | '8') << 8) | '0') << 8) | 'Y')
- #define **cmd_180Z** (((((('1' << 8) | '8') << 8) | '0') << 8) | 'Z')
- #define **cmd_M90X** (((((('M' << 8) | '9') << 8) | '0') << 8) | 'X')
- #define **cmd_P90X** (((((('P' << 8) | '9') << 8) | '0') << 8) | 'X')
- #define **cmd_M90Y** (((((('M' << 8) | '9') << 8) | '0') << 8) | 'Y')
- #define **cmd_P90Y** (((((('P' << 8) | '9') << 8) | '0') << 8) | 'Y')
- #define **cmd_M90Z** (((((('M' << 8) | '9') << 8) | '0') << 8) | 'Z')
- #define **cmd_P90Z** (((((('P' << 8) | '9') << 8) | '0') << 8) | 'Z')
- #define **cmd_PA00** (((((('P' << 8) | 'A') << 8) | '0') << 8) | '0')
- #define **cmd_PA01** (((((('P' << 8) | 'A') << 8) | '0') << 8) | '1')
- #define **cmd_PA02** (((((('P' << 8) | 'A') << 8) | '0') << 8) | '2')
- #define **cmd_PA03** (((((('P' << 8) | 'A') << 8) | '0') << 8) | '3')
- #define **cmd_PA04** (((((('P' << 8) | 'A') << 8) | '0') << 8) | '4')
- #define **cmd_PA05** (((((('P' << 8) | 'A') << 8) | '0') << 8) | '5')
- #define **cmd_PA06** (((((('P' << 8) | 'A') << 8) | '0') << 8) | '6')
- #define **cmd_PA07** (((((('P' << 8) | 'A') << 8) | '0') << 8) | '7')
- #define **cmd_PA08** (((((('P' << 8) | 'A') << 8) | '0') << 8) | '8')
- #define **cmd_PA09** (((((('P' << 8) | 'A') << 8) | '0') << 8) | '9')
- #define **cmd_PA10** (((((('P' << 8) | 'A') << 8) | '1') << 8) | '0')
- #define **cmd_PA11** (((((('P' << 8) | 'A') << 8) | '1') << 8) | '1')

Functions

- void **DecodeCommandBytes** (Globals ∗gbls, char iCommandBuffer[ ], uint8 sUART_InputBuffer[ ], uint16 nbytes)

5.13.1 Detailed Description

Command interpreter which interfaces to the Sensor Fusion Toolbox.

5.14 driver_FXAS21002.c File Reference

Provides init() and read() functions for the FXAS21002 gyroscope.

```
#include "board.h"
#include "anomaly_detection.h"
#include "sensor_drv.h"
#include "sensor_io_i2c.h"
#include "drivers.h"
#include "fxas21002.h"
```

Macros

- #define **FXAS21000_STATUS** 0x00
- #define **FXAS21000_F_STATUS** 0x08
- #define **FXAS21000_F_SETUP** 0x09
- #define **FXAS21000_WHO_AM_I** 0x0C
- #define **FXAS21000_CTRL_REG0** 0x0D
- #define **FXAS21000_CTRL_REG1** 0x13
- #define **FXAS21000_CTRL_REG2** 0x14
- #define **FXAS21000_WHO_AM_I_VALUE** 0xD1
- #define **FXAS21000_COUNTSPERDEGPERSEC** 20
- #define **FXAS21002_COUNTSPERDEGPERSEC** 16
- #define FXAS21002_GYRO_FIFO_SIZE 32

  FXAX21000, FXAS21002 have 32 element FIFO.

5.14.1 Detailed Description

Provides init() and read() functions for the FXAS21002 gyroscope.

5.15 driver_FXLS8471Q.c File Reference

Provides init() and read() functions for the FXLS8471Q 3-axis accel.

```
#include "board.h"
#include "sensor_fusion.h"
#include "sensor_drv.h"
#include "register_io_spi.h"
#include "sensor_io_spi.h"
#include "sensor_io_i2c.h"
#include "fxls8471q.h"
#include "fxls8471q_drv.h"
#include "drivers.h"
```

Macros

- #define **FXLS8471Q_COUNTSPERG** 8192.0
- #define **FXLS8471Q_ACCEL_FIFO_SIZE** 32

Functions

- int8_t **FXLS8471Q_Init** (struct PhysicalSensor ∗sensor, Globals ∗gbls)
- int8_t **FXLS8471Q_Read** (struct PhysicalSensor ∗sensor, Globals ∗gbls)
- int8_t **FXLS8471Q_Idle** (struct PhysicalSensor ∗sensor, Globals ∗gbls)

Variables

- const registerreadlist_t **FXLS8471Q_WHO_AM_I_READ** [ ]
- const registerreadlist_t **FXLS8471Q_F_STATUS_READ** [ ]
- registerreadlist_t **FXLS8471Q_DATA_READ** [ ]
- const registerwritelist_t **FXLS8471Q_Initialization** [ ]
- const registerwritelist_t **FXLS8471Q_IDLE** [ ]

5.15.1 Detailed Description

Provides init() and read() functions for the FXLS8471Q 3-axis accel.

Supports both I2C and SPI Interfaces. Supply sensor address=0x00 when when installing sensor for SPI. Supply I2C address otherwise.

5.15.2 Variable Documentation

5.15.2.1 FXLS8471Q_DATA_READ

`registerreadlist_t FXLS8471Q_DATA_READ[ ]`

Initial value:

```
=
{
    { .readFrom = FXLS8471Q_OUT_X_MSB, .numBytes = 6 }, __END_READ_DATA__
}
```

Definition at line 63 of file driver_FXLS8471Q.c.

5.15.2.2 FXLS8471Q_F_STATUS_READ

`const registerreadlist_t FXLS8471Q_F_STATUS_READ[ ]`

Initial value:

```
=
{
    { .readFrom = FXLS8471Q_F_STATUS, .numBytes = 1 }, __END_READ_DATA__
}
```

Definition at line 57 of file driver_FXLS8471Q.c.

5.15.2.3 FXLS8471Q_IDLE

```
const registerwritelist_t FXLS8471Q_IDLE[]
```

Initial value:

```
=
{
  { FXLS8471Q_CTRL_REG1, 0x00, 0x01 },
    __END_WRITE_DATA__
}
```

Definition at line 227 of file driver_FXLS8471Q.c.

5.15.2.4 FXLS8471Q_WHO_AM_I_READ

```
const registerreadlist_t FXLS8471Q_WHO_AM_I_READ[]
```

Initial value:

```
=
{
    { .readFrom = FXLS8471Q_WHO_AM_I, .numBytes = 1 }, __END_READ_DATA__
}
```

Definition at line 51 of file driver_FXLS8471Q.c.

5.16 driver_FXLS8952.c File Reference

Provides init() and read() functions for the FXLS8952 3-axis accelerometer.

```
#include "board.h"
#include "sensor_fusion.h"
#include "sensor_io_i2c.h"
#include "sensor_drv.h"
#include "FXLS8952.h"
#include "drivers.h"
```

Macros

- #define **FXLS8952_COUNTSPERG** 512
- #define **FXLS8952_ACCEL_FIFO_SIZE** 32

5.16.1 Detailed Description

Provides init() and read() functions for the FXLS8952 3-axis accelerometer.

5.17 driver_FXOS8700.c File Reference

Provides init() and read() functions for the FXOS8700 6-axis accel plus mag.

```
#include "board.h"
#include "anomaly_detection.h"
#include "sensor_io_i2c.h"
#include "fxos8700.h"
#include "fxos8700_drv.h"
#include "drivers.h"
#include "status.h"
```

Macros

- #define FXOS8700_ACCEL_FIFO_SIZE 32

    FXOS8700 (accel), MMA8652, FXLS8952 all have 32 element FIFO.
- #define FXOS8700_MAG_FIFO_SIZE 1

    FXOS8700 (mag), MAG3110 have no FIFO so equivalent to 1 element FIFO.
- #define **FXOS8700_COUNTSPERG** 8192.0
- #define **FXOS8700_COUNTSPERUT** 10

Functions

- int8_t **FXOS8700_Init** (struct PhysicalSensor ∗sensor, Globals ∗gbls)
- int8_t **FXOS8700_Read** (struct PhysicalSensor ∗sensor, Globals ∗gbls)
- int8_t **FXOS8700_Idle** (struct PhysicalSensor ∗sensor, Globals ∗gbls)

Variables

- const registerreadlist_t **FXOS8700_WHO_AM_I_READ** [ ]
- const registerreadlist_t **FXOS8700_F_STATUS_READ** [ ]
- registerreadlist_t **FXOS8700_DATA_READ** [ ]
- const registerwritelist_t **FXOS8700_Initialization** [ ]
- const registerwritelist_t **FXOS8700_FULL_IDLE** [ ]

5.17.1 Detailed Description

Provides init() and read() functions for the FXOS8700 6-axis accel plus mag.

5.17.2 Variable Documentation

5.17.2.1 FXOS8700_DATA_READ

```
registerreadlist_t FXOS8700_DATA_READ[]
```

Initial value:

```
=
{
    { .readFrom = FXOS8700_OUT_X_MSB, .numBytes = 6 }, __END_READ_DATA__
}
```

Definition at line 59 of file driver_FXOS8700.c.

5.17.2.2 FXOS8700_F_STATUS_READ

```
const registerreadlist_t FXOS8700_F_STATUS_READ[]
```

Initial value:

```
=
{
    { .readFrom = FXOS8700_STATUS, .numBytes = 1 }, __END_READ_DATA__
}
```

Definition at line 53 of file driver_FXOS8700.c.

5.17.2.3 FXOS8700_FULL_IDLE

```
const registerwritelist_t FXOS8700_FULL_IDLE[]
```

Initial value:

```
=
{

  { FXOS8700_CTRL_REG1, 0x00, 0x01 },
    __END_WRITE_DATA__
}
```

Definition at line 285 of file driver_FXOS8700.c.

5.17.2.4 FXOS8700_WHO_AM_I_READ

```
const registerreadlist_t FXOS8700_WHO_AM_I_READ[]
```

Initial value:

```
=
{
    { .readFrom = FXOS8700_WHO_AM_I, .numBytes = 1 }, __END_READ_DATA__
}
```

Definition at line 47 of file driver_FXOS8700.c.

5.18 driver_KSDK_NVM.c File Reference

middleware driver for NVM on Kinetis devices

```
#include "anomaly_detection.h"
#include "driver_KSDK_NVM.h"
#include "fsl_flash.h"
```

Macros

- #define **ERROR** 1
- #define **SUCCESS** 0;

Functions

- byte **NVM_SetBlockFlash** (uint8_t ∗Source, uint32_t Dest, uint16_t Count)

5.18.1 Detailed Description

middleware driver for NVM on Kinetis devices

5.19 driver_KSDK_NVM.h File Reference

middleware driver for NVM on Kinetis devices

Functions

- byte **NVM_SetBlockFlash** (uint8_t ∗Source, uint32_t Dest, uint16_t Count)

5.19.1 Detailed Description

middleware driver for NVM on Kinetis devices

5.20 driver_MAG3110.c File Reference

Provides init() and read() functions for the MAG3110 magnetometer.

```
#include "board.h"
#include "sensor_fusion.h"
#include "sensor_drv.h"
#include "sensor_io_i2c.h"
#include "drivers.h"
#include "mag3110.h"
```

Macros

- #define **MAG3110_COUNTSPERUT** 10

5.20.1 Detailed Description

Provides init() and read() functions for the MAG3110 magnetometer.

5.21 driver_MMA845X.c File Reference

Provides init() and read() functions for the MMA845x 3-axis accel family.

```
#include "board.h"
#include "sensor_fusion.h"
#include "sensor_drv.h"
#include "sensor_io_i2c.h"
#include "MMA845x.h"
#include "drivers.h"
```

Macros

- #define **MMA845x_COUNTSPERG** 8192.0
- #define **MMA8451_ACCEL_FIFO_SIZE** 32

5.21.1 Detailed Description

Provides init() and read() functions for the MMA845x 3-axis accel family.

Supports MMA8451Q, MMA8452Q and MMA8453Q. Key differences which are applicable to this driver are shown in the table below.

Feature	MMA8451Q	MMA8452Q	MMA8453Q
# Bits	14	12	10
FIFO	32-deep	NONE	NONE

All three have the MSB of the result registers located in the same location, so if we read as one 16-bit value, the only thing that should change g/count will be which range (+/- 2/4/8g) we are on.

5.22 driver_MMA8652.c File Reference

Provides init() and read() functions for the MMA8652 3-axis accel family.

```
#include "board.h"
#include "sensor_fusion.h"
```

```
#include "sensor_drv.h"
#include "sensor_io_i2c.h"
#include "mma865x.h"
#include "drivers.h"
```

Macros

- #define **MMA8652_COUNTSPERG** 8192.0
- #define **MMA8652_ACCEL_FIFO_SIZE** 32

5.22.1 Detailed Description

Provides init() and read() functions for the MMA8652 3-axis accel family.

1G

5.23 driver_MPL3115.c File Reference

Provides init() and read() functions for the MPL3115 pressure sensor/altimeter.

```
#include "board.h"
#include "sensor_fusion.h"
#include "sensor_io_i2c.h"
#include "mpl3115_drv.h"
#include "drivers.h"
```

Macros

- #define **MPL3115_MPERCOUNT** 0.0000152587890625F
- #define **MPL3115_CPERCOUNT** 0.00390625F
- #define **MPL3115_ACCEL_FIFO_SIZE** 32

5.23.1 Detailed Description

Provides init() and read() functions for the MPL3115 pressure sensor/altimeter.

5.24 driver_pit.c File Reference

Provides a simple abstraction for a periodic interval timer.

```
#include "issdk_hal.h"
#include "board.h"
#include "fsl_pit.h"
#include "pin_mux.h"
#include "clock_config.h"
```

Macros

- #define **PIT_LED_HANDLER** PIT0_IRQHandler
- #define **PIT_IRQ_ID** PIT0_IRQn
- #define **PIT_SOURCE_CLOCK** CLOCK_GetFreq(kCLOCK_BusClk)

Functions

- void **PIT_LED_HANDLER** (void)
- void **pit_init** (uint32_t microseconds)

Variables

- volatile bool **pitIsrFlag** = false

5.24.1 Detailed Description

Provides a simple abstraction for a periodic interval timer.

Bare metal implementations of the sensor fusion library require at least one periodic interrupt for use as a timebase for sensor fusion functions. The Periodic Interval Timer (PIT) is one such module that is commonly found on NXP Kinetis MCUs. The PIT functions are only referenced at the main() level. There is no interaction within the fusion routines themselves.

5.25 driver_pit.h File Reference

Provides a simple abstraction for a periodic interval timer.

Functions

- void **pit_init** (uint32_t microseconds)

Variables

- volatile bool **pitIsrFlag**

5.25.1 Detailed Description

Provides a simple abstraction for a periodic interval timer.

Bare metal implementations of the sensor fusion library require at least one periodic interrupt for use as a timebase for sensor fusion functions. The Periodic Interval Timer (PIT) is one such module that is commonly found on NXP Kinetis MCUs. The PIT functions are only referenced at the main() level. There is no interaction within the fusion routines themselves.

5.26 driver_systick.c File Reference

Encapsulates the ARM sysTick counter, which is used for benchmarking.

```
#include "anomaly_detection.h"
#include "drivers.h"
```

Macros

- #define **SYST_CSR** SysTick->CTRL
- #define **SYST_RVR** SysTick->LOAD
- #define **SYST_CVR** SysTick->VAL

Functions

- void **ARM_systick_enable** (void)
- void **ARM_systick_start_ticks** (int32 *pstart)
- int32 **ARM_systick_elapsed_ticks** (int32 start_ticks)
- void **ARM_systick_delay_ms** (uint32 iSystemCoreClock, uint32 delay_ms)

5.26.1 Detailed Description

Encapsulates the ARM sysTick counter, which is used for benchmarking.

5.27 drivers.h File Reference

Provides function prototypes for driver level interfaces.

```
#include "Driver_I2C.h"
#include "Driver_SPI.h"
```

Functions

SysTick Macros

The ARM SysTick counter is used to time various fusion options. Timings are then conveyed to the NXP Sensor Fusion Toolbox, where they are displayed for the developer. These functions should be portable to any ARM M0+, M3, M4 or M4F device. If you are using a different CPU architecture, you will need to provide an equivalent set of macros, remove the macro calls from the fusion routines, or define a set of empty macros.

- void **ARM_systick_enable** (void)
- void **ARM_systick_start_ticks** (int32_t *pstart)
- int32_t **ARM_systick_elapsed_ticks** (int32_t start_ticks)
- void **ARM_systick_delay_ms** (uint32_t iSystemCoreClock, uint32_t delay_ms)

Sensor Drivers

Each physical sensor must be provided with one initialization function and one "read" function. These must be installed by the user using the installSensor method defined in Globals. By "physical sensor", we mean either individual sensor type (such as a 3-axis accelerometer) or a combo-sensor such as the NXP FXOS8700 6-axis accel plus mag. The init() function for each sensor is responsible for initializing all sensors contained in that package. The read() function is responsible for reading those same sensors and moving the results into the standard structures contained within the Globals object.

- int8_t **MPL3115_Init** (struct PhysicalSensor ∗sensor, Globals ∗gbls)
- int8_t **FXOS8700_Init** (struct PhysicalSensor ∗sensor, Globals ∗gbls)
- int8_t **FXAS21002_Init** (struct PhysicalSensor ∗sensor, Globals ∗gbls)
- int8_t **MMA8652_Init** (struct PhysicalSensor ∗sensor, Globals ∗gbls)
- int8_t **FXLS8952_Init** (struct PhysicalSensor ∗sensor, Globals ∗gbls)
- int8_t **MAG3110_Init** (struct PhysicalSensor ∗sensor, Globals ∗gbls)
- int8_t **MMA8451_Init** (struct PhysicalSensor ∗sensor, Globals ∗gbls)
- int8_t **FXLS8471Q_Init** (struct PhysicalSensor ∗sensor, Globals ∗gbls)
- int8_t **MPL3115_Read** (struct PhysicalSensor ∗sensor, Globals ∗gbls)
- int8_t **FXOS8700_Read** (struct PhysicalSensor ∗sensor, Globals ∗gbls)
- int8_t **FXAS21002_Read** (struct PhysicalSensor ∗sensor, Globals ∗gbls)
- int8_t **MMA8652_Read** (struct PhysicalSensor ∗sensor, Globals ∗gbls)
- int8_t **FXLS8952_Read** (struct PhysicalSensor ∗sensor, Globals ∗gbls)
- int8_t **MAG3110_Read** (struct PhysicalSensor ∗sensor, Globals ∗gbls)
- int8_t **MMA8451_Read** (struct PhysicalSensor ∗sensor, Globals ∗gbls)
- int8_t **FXLS8471Q_Read** (struct PhysicalSensor ∗sensor, Globals ∗gbls)
- int8_t **MPL3115_Idle** (struct PhysicalSensor ∗sensor, Globals ∗gbls)
- int8_t **FXOS8700_Idle** (struct PhysicalSensor ∗sensor, Globals ∗gbls)
- int8_t **FXAS21002_Idle** (struct PhysicalSensor ∗sensor, Globals ∗gbls)
- int8_t **MMA8652_Idle** (struct PhysicalSensor ∗sensor, Globals ∗gbls)
- int8_t **FXLS8952_Idle** (struct PhysicalSensor ∗sensor, Globals ∗gbls)
- int8_t **MAG3110_Idle** (struct PhysicalSensor ∗sensor, Globals ∗gbls)
- int8_t **MMA8451_Idle** (struct PhysicalSensor ∗sensor, Globals ∗gbls)
- int8_t **FXLS8471Q_Idle** (struct PhysicalSensor ∗sensor, Globals ∗gbls)

5.27.1 Detailed Description

Provides function prototypes for driver level interfaces.

Users who are not using NXP hardware will need to supply their own drivers in place of those defined here.

5.28 fusion.c File Reference

Lower level sensor fusion interface.

```
#include "stdio.h"
#include "math.h"
#include "stdlib.h"
#include "anomaly_detection.h"
#include "fusion.h"
#include "orientation.h"
#include "matrix.h"
#include "approximations.h"
#include "drivers.h"
#include "control.h"
```

Functions

- void **fInitializeFusion** (Globals *gbls)
- void **fFuseSensors** (struct SV_1DOF_P_BASIC *pthisSV_1DOF_P_BASIC, struct SV_3DOF_G_BASIC *pthisSV_3DOF_G_BASIC, struct SV_3DOF_B_BASIC *pthisSV_3DOF_B_BASIC, struct SV_3DOF_Y_↩ BASIC *pthisSV_3DOF_Y_BASIC, struct SV_6DOF_GB_BASIC *pthisSV_6DOF_GB_BASIC, struct SV_↩ 6DOF_GY_KALMAN *pthisSV_6DOF_GY_KALMAN, struct SV_9DOF_GBY_KALMAN *pthisSV_9DOF_↩ GBY_KALMAN, struct AccelSensor *pthisAccel, struct MagSensor *pthisMag, struct GyroSensor *pthisGyro, struct PressureSensor *pthisPressure, struct MagCalibration *pthisMagCal)
- void **fInit_1DOF_P_BASIC** (struct SV_1DOF_P_BASIC *pthisSV, struct PressureSensor *pthisPressure, float flpftimesecs)
- void **fInit_3DOF_G_BASIC** (struct SV_3DOF_G_BASIC *pthisSV, struct AccelSensor *pthisAccel, float flpf-timesecs)
- void **fInit_3DOF_B_BASIC** (struct SV_3DOF_B_BASIC *pthisSV, struct MagSensor *pthisMag, float flpf-timesecs)
- void **fInit_3DOF_Y_BASIC** (struct SV_3DOF_Y_BASIC *pthisSV)
- void **fInit_6DOF_GB_BASIC** (struct SV_6DOF_GB_BASIC *pthisSV, struct AccelSensor *pthisAccel, struct MagSensor *pthisMag, float flpftimesecs)
- void **fInit_6DOF_GY_KALMAN** (struct SV_6DOF_GY_KALMAN *pthisSV, struct AccelSensor *pthisAccel, struct GyroSensor *pthisGyro)
- void **fInit_9DOF_GBY_KALMAN** (struct SV_9DOF_GBY_KALMAN *pthisSV, struct AccelSensor *pthis↩ Accel, struct MagSensor *pthisMag, struct GyroSensor *pthisGyro, struct MagCalibration *pthisMagCal)
- void **fRun_1DOF_P_BASIC** (struct SV_1DOF_P_BASIC *pthisSV, struct PressureSensor *pthisPressure)
- void **fRun_3DOF_G_BASIC** (struct SV_3DOF_G_BASIC *pthisSV, struct AccelSensor *pthisAccel)
- void **fRun_3DOF_B_BASIC** (struct SV_3DOF_B_BASIC *pthisSV, struct MagSensor *pthisMag)
- void **fRun_3DOF_Y_BASIC** (struct SV_3DOF_Y_BASIC *pthisSV, struct GyroSensor *pthisGyro)
- void **fRun_6DOF_GB_BASIC** (struct SV_6DOF_GB_BASIC *pthisSV, struct MagSensor *pthisMag, struct AccelSensor *pthisAccel)
- void **fRun_6DOF_GY_KALMAN** (struct SV_6DOF_GY_KALMAN *pthisSV, struct AccelSensor *pthisAccel, struct GyroSensor *pthisGyro)

5.28.1 Detailed Description

Lower level sensor fusion interface.

5.29 fusion.h File Reference

Lower level sensor fusion interface.

```
#include "anomaly_detection.h"
```

Macros

COMPUTE_1DOF_P_BASIC constants

- #define FLPFSECS_1DOF_P_BASIC 1.5F
  pressure low pass filter time constant (s)

COMPUTE_3DOF_G_BASIC constants

- #define FLPFSECS_3DOF_G_BASIC 1.0F
  tilt orientation low pass filter time constant (s)

COMPUTE_3DOF_B_BASIC constants

- #define FLPFSECS_3DOF_B_BASIC 7.0F
  2D eCompass orientation low pass filter time constant (s)

COMPUTE_6DOF_GB_BASIC constants

- #define **FLPFSECS_6DOF_GB_BASIC** 7.0F

COMPUTE_6DOF_GY_KALMAN constants

- #define FQVY_6DOF_GY_KALMAN 2E2
  gyro sensor noise variance units (deg/s)^2
- #define FQVG_6DOF_GY_KALMAN 1.2E-3
  accelerometer sensor noise variance units g^2
- #define FQWB_6DOF_GY_KALMAN 2E-2F
  gyro offset random walk units (deg/s)^2
- #define FMIN_6DOF_GY_BPL -7.0F
  minimum permissible power on gyro offsets (deg/s)
- #define FMAX_6DOF_GY_BPL 7.0F
  maximum permissible power on gyro offsets (deg/s)

COMPUTE_9DOF_GBY_KALMAN constants

gyro sensor noise covariance units deg^2 increasing this parameter improves convergence to the geomagnetic field

- #define FQVY_9DOF_GBY_KALMAN 2E2
  gyro sensor noise variance units (deg/s)^2
- #define FQVG_9DOF_GBY_KALMAN 1.2E-3
  accelerometer sensor noise variance units g^2 defining minimum deviation from 1g sphere
- #define FQVB_9DOF_GBY_KALMAN 5E0
  magnetometer sensor noise variance units uT^2 defining minimum deviation from geomagnetic sphere.
- #define FQWB_9DOF_GBY_KALMAN 2E-2F
  gyro offset random walk units (deg/s)^2
- #define FMIN_9DOF_GBY_BPL -7.0F
  minimum permissible power on gyro offsets (deg/s)
- #define FMAX_9DOF_GBY_BPL 7.0F
  maximum permissible power on gyro offsets (deg/s)

Functions

Fusion Function Prototypes

These functions comprise the core of the basic sensor fusion functions excluding magnetic and acceleration calibration. Parameter descriptions are not included here, as details are provided in anomaly_detection.h.

- void **fInitializeFusion** (Globals *gbls)

5.29.1 Detailed Description

Lower level sensor fusion interface.

This file can be used to "tune" the performance of specific algorithms within the sensor fusion library. It also defines the lower level function definitions for specific algorithms. Normally, the higher level hooks in anomaly_detection.h will be used, and those shown here will be left alone.

5.30 hal_frdm_fxs_mult2_b.c File Reference

Hardware Abstraction layer for the FRDM-FXS-MULT2-B sensor shield.

```
#include "sensor_fusion.h"
```

Functions

- void ApplyAccelHAL (struct AccelSensor ∗Accel)
  Apply the accelerometer Hardware Abstraction Layer.
- void ApplyMagHAL (struct MagSensor ∗Mag)
  Apply the magnetometer Hardware Abstraction Layer.
- void ApplyGyroHAL (struct GyroSensor ∗Gyro)
  Apply the gyroscope Hardware Abstraction Layer.

5.30.1 Detailed Description

Hardware Abstraction layer for the FRDM-FXS-MULT2-B sensor shield.

5.30.2 Function Documentation

5.30.2.1 ApplyAccelHAL()

```
void ApplyAccelHAL (
            struct AccelSensor * Accel )
```

Apply the accelerometer Hardware Abstraction Layer.

Parameters

Accel	pointer to accelerometer logical sensor

Definition at line 44 of file hal_frdm_fxs_mult2_b.c.

5.30.2.2 ApplyGyroHAL()

```
void ApplyGyroHAL (
            struct GyroSensor * Gyro )
```

Apply the gyroscope Hardware Abstraction Layer.

Parameters

Gyro	pointer to gyroscope logical sensor

Definition at line 99 of file hal_frdm_fxs_mult2_b.c.

5.30.2.3 ApplyMagHAL()

```
void ApplyMagHAL (
            struct MagSensor * Mag )
```

Apply the magnetometer Hardware Abstraction Layer.

Parameters

Mag	pointer to magnetometer logical sensor

Definition at line 71 of file hal_frdm_fxs_mult2_b.c.

5.31 machineLearning_subsystem.c File Reference

The machinelearning_subsystem.c file implements the top level programming interface.

```
#include <stdio.h>
#include <math.h>
#include "anomaly_detection.h"
#include "status.h"
#include "magnetic.h"
#include "drivers.h"
#include "sensor_drv.h"
#include "control.h"
#include "fusion.h"
#include "fsl_debug_console.h"
#include "machineLearning_subsystem.h"
#include "gmm_utility.h"
```

Functions

- void initFeatureCollection (struct FeatureCollection *featureCollection, struct FeatureInstance *featureInstance)

  Initializes the FeatureCollection structure.
- void initModelCollection (struct ModelCollection *modelCollection, struct ModelInstance *modelInstance, union Model *model)

  Initializes the ModelCollection structure.
- void disableAllModels (struct ModelCollection *models)

  Stops all the active models.
- int8_t **idxViaFindOrAssignModelID** (struct ModelCollection *models, uint8_t modelID)
- void addToFeatureBuffers (struct Globals *gbls)

  Adds the real-time features to the buffer in FeatureInstance.
- void addToFeatBuffer (struct Globals *gbls, struct FeatureInstance *featureInstance)

  Adds a feature sample to the fBuffer.
- void zscoreFeatBuffer (struct FeatureInstance *featureInstance)

  Computes zscore (standardized by the given std and mean).
- void computeZscore (float *xn, float x, float mu, float sigma)

  Computes zscore.
- void incrementFeatureBufferIndices (struct Globals *gbls)

  increases the featuer buffer indices.
- void incrementFeatureBufferIndicesModels (struct Globals *gbls)

  Increment Feature Buffer Indices of Models.
- void incrementFeatureBufferIndicesStartEnd (struct ModelInstance *modelInstance)

  Increases feature buffer indices for a model instance.
- bool checkReversed (int startPosition, int endPosition)

  Checks whether a moving window is reversed or not.
- void inputPositions (struct ModelInstance *modelInstance, int *startPos, int *endPos)

  Finds a window position.
- float **getLastFeatureValueAdded** (FeatureInstance *featureInstance)
- void normalizeFeatures (struct ModelInstance *modelInstance)

  Normalizes the features within a moving window.
- void normalizeFeatWindow (struct FeatureInstance *featureInstance, int sPos, int ePos, bool reversed)
- bool get_model_idx_instance (uint8_t *idx, bool *isEnabled, struct ModelCollection *modelCollection, uint8_t modelID)

  Gets the model instance by model ID.
- bool get_feature_number (uint8_t *idx, struct FeatureCollection *featureCollection, sensor_t sensor, axis_t axis, feature_t feature)

  Gets feature instance number.
- bool add_feature (uint8_t *idx, struct FeatureCollection *featureCollection, sensor_t sensor, axis_t axis, feature_t feature)

  Adds features by (sensor_t, axis_t, feature_t).
- bool **delete_feature_by_index** (struct FeatureCollection *features, uint8_t idx)
- bool **delete_feature_by_attributes** (struct FeatureCollection *features, sensor_t sensor, axis_t axis, feature_t feature)
- bool **get_feature_attributes** (struct FeatureCollection *features, uint8_t idx, sensor_t *sensor, axis_t *axis, feature_t *feature)
- bool enableRunModel (struct ModelCollection *modelCollection, uint8_t modelID)

  Enbales a model with model ID to run.
- bool enableModel (struct ModelCollection *modelCollection, uint8_t modelID)

  Enables model instances.
- bool runModel (struct ModelInstance *modelInstance, bool *passFail, float *likelihood)

Execute RUN mode of a model instance.
- void detectAnomalyInGMM (struct ModelInstance ∗modelInstance)

  Anomaly detection function for GMM.
- void detectAnomalyInSVM (struct ModelInstance ∗modelInstance)

  Anomaly detection function for SVM.
- void detectAnomalyInEnsemble (struct ModelInstance ∗modelInstance)

  Anomaly detection function for Ensemble Models.
- bool trainModel (struct ModelCollection ∗modelCollection, uint8_t modelID)

  Executes TRAINING of model instances.
- void **trainGMM_instance** (struct ModelInstance ∗modelInstance)
- void trainGMM (struct ModelCollection ∗modelCollection, uint8_t modelID)

  Executes TRAINING of GMM instance.
- float **sumBuffer** (float ∗pBuffer, int startPos, int endPos, bool reversed)
- float **sumSquaredBuffer** (float ∗pBuffer1, float ∗pBuffer2, int startPos, int endPos, bool reversed)
- bool **computeR** (struct ModelInstance ∗modelInstance, float R[ ][2])
- bool initGMM (struct ModelInstance ∗modelInstance)

  Initializes GMM parameters.
- void runGMMEM (struct ModelInstance ∗modelInstance)

  Executes expectation-maximization (EM) algorithm to estimate GMM parameters.
- void refreshGMM (struct ModelInstance ∗modelInstance)

  Refreshes GMM parameters.
- void **trainOCSVM_instance** (struct ModelInstance ∗modelInstance)
- void trainOCSVM (struct ModelCollection ∗modelCollection, uint8_t modelID)

  Executes TRAINING of SVM instance.
- void trainOCSVM_malloc (struct ModelInstance ∗modelInstance)

  Executes OC-SVM algorithm to estimate SVs.
- void save_model_instance (struct svm_model ∗model, struct OcsvmModel ∗ocsvm)

  To save the trained SVM model into a ModelInstance structure.
- void parse_svm_param (struct OcsvmModel ∗ocsvm, struct svm_parameter ∗param)

  To parse the input parameters for SVM before training.
- void **trainEnsemble** (struct ModelCollection ∗modelCollection)
- void **clearFeatureBuffers_Instance** (struct ModelInstance ∗modelInstance)

Variables

- Globals gbls

  This is the primary sensor fusion data structure.

5.31.1 Detailed Description

The machinelearning_subsystem.c file implements the top level programming interface.

This contains subroutines of machine learning algorithms.

5.31.2 Function Documentation

5.31.2.1 add_feature()

```
bool add_feature (
            uint8_t * idx,
            struct FeatureCollection * featureCollection,
            sensor_t sensor,
            axis_t axis,
            feature_t feature )
```

Adds features by (sensor_t, axis_t, feature_t).

This function is to add and enable feature instance with coordination (sensor_t, axis_t, feature_t).

Parameters

idx	The output that indicates the resulting index of featureInstance.
featureCollection	The pointer of FeatureCollection structure.
sensor	Sensor type information.
axis	Sensor axis information.
feature	Feature type information.

Returns

true / false.

Definition at line 526 of file machineLearning_subsystem.c.

5.31.2.2 addToFeatBuffer()

```
void addToFeatBuffer (
            struct Globals * gbls,
            struct FeatureInstance * featureInstance )
```

Adds a feature sample to the fBuffer.

Feature Operation Functions control functions for machine learning feature operation for creating and saving features from raw sensor data.

This function adds a feature sample to the feature buffer in feature instance. The Globals structure contains four different types sensors where each feature instance takes feature samples from only one of the sensors. Definitions - F_USING_ACCEL, F_USING_MAG, F_USING_GYRO, and F_1DOF_P_BASIC separate the sensor types.

Parameters

gbls	The pointer of Globals structure.
featureInstance	The pointer of FeatureInstance structure.

Returns

void

Definition at line 196 of file machineLearning_subsystem.c.

5.31.2.3 addToFeatureBuffers()

```
void addToFeatureBuffers (
            struct Globals * gbls )
```

Adds the real-time features to the buffer in FeatureInstance.

This function is to add the current features computed in real time into the buffer of FeatureInstance. It first finds which feature instance is enabled, and then calls addToFeatBuffer() function in order to put the values into the enabled feature instance.

Parameters

gbls	The pointer of Globals structure.

Returns

void.

Definition at line 169 of file machineLearning_subsystem.c.

5.31.2.4 checkReversed()

```
bool checkReversed (
            int startPosition,
            int endPosition )
```

Checks whether a moving window is reversed or not.

This function is to check whether a moving window is reversed or not. If the window is reversed, then the start↔Position will be bigger than endPosition. If not, startPosition will be smaller than endPosition.

Parameters

startPosition	The index that indicates the start position of a moving window.
endPosition	The index that indicates the end position of a moving window.

Returns

true / false.

Definition at line 342 of file machineLearning_subsystem.c.

5.31.2.5 computeZscore()

```
void computeZscore (
            float * xn,
            float x,
            float mu,
            float sigma )
```

Computes zscore.

This function computes zscore instantly.

Parameters

xn	The pointer for the output of this function.
x	The feature to be standardized.
mu	Mean.
sigma	Standard deviation.

Returns

void.

Definition at line 254 of file machineLearning_subsystem.c.

5.31.2.6 detectAnomalyInEnsemble()

```
void detectAnomalyInEnsemble (
            struct ModelInstance * modelInstance )
```

Anomaly detection function for Ensemble Models.

This function is for anomaly detection using ensemble models. Soft and hard decisions are made.

Parameters

modelInstance	The pointer of ModelInstance structure.

Returns

void.

Definition at line 743 of file machineLearning_subsystem.c.

5.31.2.7 detectAnomalyInGMM()

```
void detectAnomalyInGMM (
            struct ModelInstance * modelInstance )
```

Anomaly detection function for GMM.

This function is for anomaly detection using GMM. Soft and hard decisions are made.

Parameters

modelInstance	The pointer of ModelInstance structure.

Returns

void.

Definition at line 669 of file machineLearning_subsystem.c.

5.31.2.8 detectAnomalyInSVM()

```
void detectAnomalyInSVM (
            struct ModelInstance * modelInstance )
```

Anomaly detection function for SVM.

This function is for anomaly detection using OC-SVM. Soft and hard decisions are made.

Parameters

modelInstance	The pointer of ModelInstance structure.

Returns

void.

Definition at line 699 of file machineLearning_subsystem.c.

5.31.2.9 disableAllModels()

```
void disableAllModels (
            struct ModelCollection * models )
```

Stops all the active models.

This function stops the active models by setting isEnabled to false at each model instance.

Parameters

modelCollection	The pointer of ModelCollection structure.

Returns

void.

Definition at line 136 of file machineLearning_subsystem.c.

5.31.2.10 enableModel()

```
bool enableModel (
            struct ModelCollection * modelCollection,
            uint8_t modelID )
```

Enables model instances.

This function enables modelInstance with model ID.

Parameters

modelCollection	The pointer of ModelCollection structure.
modelID	Model ID.

Returns

true / false True if success.

Definition at line 608 of file machineLearning_subsystem.c.

5.31.2.11 enableRunModel()

```
bool enableRunModel (
            struct ModelCollection * modelCollection,
            uint8_t modelID )
```

Enbales a model with model ID to run.

This function enables a model that is specified by model ID to run. iStage of modelInstance is set to RUN_COM↩ MAND, if the model instance has been trained.

Parameters

modelCollection	The pointer of ModelCollection structure.
modelID	Model ID.

Returns

true / false True if success.

Definition at line 587 of file machineLearning_subsystem.c.

5.31.2.12 get_feature_number()

```
bool get_feature_number (
            uint8_t * idx,
            struct FeatureCollection * featureCollection,
            sensor_t sensor,
            axis_t axis,
            feature_t feature )
```

Gets feature instance number.

This function gets the index of FeatureInstance structure from the coordination of sensor_type, sensor_axis, and feature_type.

Parameters

idx	The output that indicates the resulting index of featureInstance.
featureCollection	The pointer of FeatureCollection structure.
sensor	Sensor type information.
axis	Sensor axis information.
feature	Feature type information.

Returns

true / false.

Definition at line 499 of file machineLearning_subsystem.c.

5.31.2.13 get_model_idx_instance()

```
bool get_model_idx_instance (
            uint8_t * idx,
            bool * isEnabled,
            struct ModelCollection * modelCollection,
            uint8_t modelID )
```

Gets the model instance by model ID.

Model Operation Functions control functions for machine learning models operation such as training, testing, etc.

This function searches model IDs of modelInstance structures and assigns the ID number and isEnabled.

Parameters

idx	The output that indicates modelInstance index.
isEnabled	The output that indicates whether the modelInstance is enabled or not.
modelCollection	The pointer of ModelCollection structure
modelID	Model ID.

Returns

true / false True if success.

Definition at line 476 of file machineLearning_subsystem.c.

5.31.2.14 incrementFeatureBufferIndices()

```
void incrementFeatureBufferIndices (
            struct Globals * gbls )
```

increases the featuer buffer indices.

This function increases the feature buffer indices, if the feature instance is enabled. When the index exceeds M↩
AX_FEATURE_SAMPLES which equivalently defines the maximum size of buffer, the index is reset and creases again. iBufferExceeded indicates whether this occured or not.

Parameters

gbls	The pointer of Globals structure.

Returns

void

Definition at line 268 of file machineLearning_subsystem.c.

5.31.2.15 incrementFeatureBufferIndicesModels()

```
void incrementFeatureBufferIndicesModels (
            struct Globals * gbls )
```

Increment Feature Buffer Indices of Models.

This function increase the indices of feature buffers, if the model instance is enabled.

Parameters

gbls	The pointer of Globals structure.

Returns

void.

Definition at line 293 of file machineLearning_subsystem.c.

5.31.2.16 incrementFeatureBufferIndicesStartEnd()

```
void incrementFeatureBufferIndicesStartEnd (
            struct ModelInstance * modelInstance )
```

Increases feature buffer indices for a model instance.

This function increases feature buffer indices for the given model instance. It checks whether the indices are within a moving window that is reversed or not.

Parameters

modelInstance	The pointer of ModelInstance structure.

Returns

void.

Definition at line 311 of file machineLearning_subsystem.c.

5.31.2.17 initFeatureCollection()

```
void initFeatureCollection (
            struct FeatureCollection * featureCollection,
            struct FeatureInstance * featureInstance )
```

Initializes the FeatureCollection structure.

The FeatureCollection structure contains pointers of feature instances. This function initializes the address of feature instances created, sensor type, sensor axis, feature type, buffer counter, and indicators such as iBufferExceeded, isEnabled, and isNormalized.

Parameters

features	The pointer of FeatureCollection structure.
featureInstance	The first pointer of FeatureInstance array structure (if multiple feature instances are defined).

Returns

void.

Definition at line 63 of file machineLearning_subsystem.c.

5.31.2.18 initGMM()

```
bool initGMM (
            struct ModelInstance * modelInstance )
```

Initializes GMM parameters.

This function initializes GMM parameters. At every training time, this function is called to refresh the GMM parameters. If opt_num_components is less than init_maxGaussComponents, GMM increases its own components and then refresh the parameters. This way helps to smoothly adapt GMM over time.

Parameters

modelInstance	The pointer of ModelInstance structure.

Returns

true / false True if success.

Definition at line 925 of file machineLearning_subsystem.c.

5.31.2.19 initModelCollection()

```
void initModelCollection (
            struct ModelCollection * modelCollection,
            struct ModelInstance * modelInstance,
            union Model * model )
```

Initializes the ModelCollection structure.

The ModelCollection is initialized with the number of models that are going to process, training rate, and model↵ Instance structures. The ModelInstance structure contains pointers of feature instances. This function initializes the address of model instances created, feature_dimension, model ID, decision results (soft / hard), and indicators such as isEnabled and isTrained. Each modelInstance also contains startPositionBuffer and endPositionBuffer that are used for a moving window within a feature buffer. That moving window is to pick a given number of latest features from the buffer in real time. The ModelInstance structure also contains the pointers of feature instances and union of model types (GMM / SVM). Each modelInstance takes only one of GMM and SVM models.

Parameters

modelCollection	The pointer of ModelCollection structure.
modelInstance	The pointer of ModelInstance structure.
model	The pointer of union Model that contains struct GmmModel and struct OcsvmModel.

Returns

void.

Definition at line 101 of file machineLearning_subsystem.c.

5.31.2.20 inputPositions()

```
void inputPositions (
            struct ModelInstance * modelInstance,
            int * startPos,
            int * endPos )
```

Finds a window position.

This function finds the start and end positions of a moving window.

Parameters

modelInstance	The pointer of ModelInstance structure.
startPos	The output that contains the start position of window.
endPos	The output that contains the end position of window.

Returns

void.

Definition at line 357 of file machineLearning_subsystem.c.

5.31.2.21 normalizeFeatures()

```
void normalizeFeatures (
            struct ModelInstance * modelInstance )
```

Normalizes the features within a moving window.

This function is to normalize the features within a moving window whose start and end positions are maintained in ModelInstance structure. The function contains a subroutine for checking whether those positions are reversed or not. normalizedFeatWindow() is referred to.

Parameters

modelInstance	The pointer of ModelInstance structure.

Returns

void.

Definition at line 388 of file machineLearning_subsystem.c.

5.31.2.22 normalizeFeatWindow()

```
void normalizeFeatWindow (
            struct FeatureInstance * featureInstance,
            int sPos,
            int ePos,
            bool reversed )
```

Normalizes features within a moving window.

```
This function is to normalize the features within a moving window. The start and end poistions of the window a
and stored in FeatureInstance structure.
isNormalized is set to true at the end of this function.

@param  featureInstance The pointer of FeatureInstance structure.
@param  sPos    The start position of moving window.
@param  ePos    The end position of moving window.
@param  reversed        The indicator whether the moving window is reversed or not.
@return void.
```

Definition at line 421 of file machineLearning_subsystem.c.

5.31.2.23 parse_svm_param()

```
void parse_svm_param (
            struct OcsvmModel * ocsvm,
            struct svm_parameter * param )
```

To parse the input parameters for SVM before training.

This function is to parse the input parameters for SVM before training.

Parameters

ocsvm	The pointer of struct OcsvmModel which is a union of Model in ModelInstance structure.
param	The pointer of struct svm_parameter which is defined in LIBSVM.

Returns

void.

Definition at line 1210 of file machineLearning_subsystem.c.

5.31.2.24 refreshGMM()

```
void refreshGMM (
            struct ModelInstance * modelInstance )
```

Refreshes GMM parameters.

This function refreshes GMM parameters. Somtimes, GMM parameters are ill conditioned. This function is called to refresh the GMM parameters. If opt_num_components is less than init_maxGaussComponents, GMM increases its own components and then refresh the parameters. This way helps to smoothly adapt GMM over time.

Parameters

modelInstance	The pointer of ModelInstance structure.

Returns

true / false True if success.

Definition at line 1036 of file machineLearning_subsystem.c.

5.31.2.25 runGMMEM()

```
void runGMMEM (
            struct ModelInstance * modelInstance )
```

Executes expectation-maximization (EM) algorithm to estimate GMM parameters.

This function executes EM algorithm to estimate GMM parameters. With an initial set of parameters, it first computes minimum description length (MDL) criterion, which is used to find the optimal number of Gaussian components, by computeMDL_GMM(). Reducing the number of mixture components upto one, it evaluates MDL for different number of mixture components. When it reduces the number of components, reduceModelOrder() sums two adjacent componets and discard one of the two. The adjacency is measured by mean vectors of every given pairs of components. For every case of mixture components, EM algoritm is processed by calling reestimate() and computeLL_regroup() in computeMDL_GMM().

Parameters

modelInstance	The pointer of ModelInstance structure.

Returns

void.

Definition at line 984 of file machineLearning_subsystem.c.

5.31.2.26 runModel()

```
bool runModel (
            struct ModelInstance * modelInstance,
            bool * passFail,
            float * likelihood )
```

Execute RUN mode of a model instance.

This function executes the run mode of a model instance. Depending on the model type (GMM / SVM), given the trained model, detectionAnomalyInGMM(), detectionAnomalyInSVM(), or detectionAnomalyInEnsemble() is called.

Parameters

modelInstance	The pointer of ModelInstance structure.
passFail	Hard decision restult. Pass: positive (normal). Fail: negative (anomaly).
likelihood	Soft decition restult. The output of density function built by GMM or SVM.

Returns

true / false True if success.

Definition at line 628 of file machineLearning_subsystem.c.

5.31.2.27 save_model_instance()

```
void save_model_instance (
            struct svm_model * model,
            struct OcsvmModel * ocsvm )
```

To save the trained SVM model into a ModelInstance structure.

This function is to save the trained SVM model by LIBSVM into a ModelInstance structure. Type conversion is done.

Parameters

model	The pointer of struct svm_model.
ocsvm	The pointer of struct OcsvmModel which is a union of Model in ModelInstance structure.

Returns

void.

Definition at line 1179 of file machineLearning_subsystem.c.

5.31.2.28 trainGMM()

```
void trainGMM (
            struct ModelCollection * modelCollection,
            uint8_t modelID )
```

Executes TRAINING of GMM instance.

This function is for training a GMM instance. If the model instance is enabled and the features are normalized, then Expectation-Maximization (EM) algorithm runs. initGMM() is called to control GMM parameters before EM is processed. runGMMEM() executes the EM algorithm. Before calling those functions, it is assumed to check whether the moving window of feature buffer is reversed or not. Once TRAINING is done, isTrained is set to true.

Parameters

modelCollection	The pointer of ModelCollection structure.
modelID	Model ID.

Returns

true / false True if success.

Definition at line 823 of file machineLearning_subsystem.c.

5.31.2.29 trainModel()

```
bool trainModel (
            struct ModelCollection * modelCollection,
            uint8_t modelID )
```

Executes TRAINING of model instances.

This function is for training a model instance. Each modelInstance contains a model type (GMM or SVM or Ensemble), and calls one of the training functions (trainGMM(), trainOCSVM(), and trainALL()). Before calling those functions, it is assumed to normalize features because normalization can help models to avoid undesired situations (e.g., sigularity of covariance matrix in GMM). Also, normalization scales down arbitrary features down to standardized ones by std and mean.

Parameters

modelCollection	The pointer of ModelCollection structure.
modelID	Model ID.

Returns

true / false True if success.

Definition at line 757 of file machineLearning_subsystem.c.

5.31.2.30 trainOCSVM()

```
void trainOCSVM (
            struct ModelCollection * modelCollection,
            uint8_t modelID )
```

Executes TRAINING of SVM instance.

This function is for training a SVM instance. If the model instance is enabled and the features are normalized, then one class support vector machine (OC-SVM) algorithm runs. Before calling this function, it is assumed to check whether the moving window of feature buffer is reversed or not. Once TRAINING is done, isTrained is set to true.

Parameters

modelCollection	The pointer of ModelCollection structure.
modelID	Model ID.

Returns

true / false True if success.

Definition at line 1103 of file machineLearning_subsystem.c.

5.31.2.31 trainOCSVM_malloc()

```
void trainOCSVM_malloc (
            struct ModelInstance * modelInstance )
```

Executes OC-SVM algorithm to estimate SVs.

This function is for execution of OC-SVM. As a result of the algorithm, a set of support vectors (SVs) are provided, which build SVM model. The parameter nu controls the bounds on the number of SVs and fraction of anomaly in training data set. LIBSVM library is utilized to build the SVM model. Since LIBSVM uses dynamic memory allocation, this function follows the methodology by using the capability provided by FreeRTOS. The dynamic memory allocation here is based on heap_4.c. For the embedded system we are working on, we limited the maximum size of feature buffers, which means the maximum number of SVs is also limited. This may reduce the risk of use of dynamic memory allocation in the current implementation.

Parameters

modelInstance	The pointer of ModelInstance structure.

Returns

void.

Definition at line 1132 of file machineLearning_subsystem.c.

5.31.2.32 zscoreFeatBuffer()

```
void zscoreFeatBuffer (
            struct FeatureInstance * featureInstance )
```

Computes zscore (standardized by the given std and mean).

This function normalizes the feature sample in fBuffer and save it in fBufferNormalized. iBufferCount is the current index of the feature.

Parameters

featureInstance	The pointer of FeatureInstance structure.

Returns

void

Definition at line 237 of file machineLearning_subsystem.c.

5.32 machineLearning_subsystem.h File Reference

Data structures and function prototypes for the machine learning library.

```
#include "svm.h"
#include "FreeRTOS.h"
```

Data Structures

- struct FeatureInstance

  Feature Instance structure contains the buffers for feature samples and other information.
- struct FeatureCollection

  Feature Collection structure contains the poinsters of FeatureInstance structures.
- struct GaussComponent

  A Gaussian component structure in the mixture model.
- struct GmmModel

  A Gaussian mixture model (GMM) structure.
- struct svm_node_embedded

  A structure for each support vector.
- struct OcsvmModel

  A One Class Support Vector Machine (OC-SVM) structure.
- union Model

  A Union type model structure.
- struct ModelInstance

  Model Instance Structure contains the buffers for feature samples and other information.
- struct ModelCollection

  Model Collection structure contains the poinsters of ModelInstance structures.

Macros

- #define MAX_GMM 4

  Maximum number of Gaussian Mixture Model allowed to train/run in the embedded system.
- #define MAX_GAUSSIAN_COMPONENTS 7

  Maximum number of Gaussian components allowed for a single GMM.
- #define MAX_FEATURE_DIMENSION 2

  Maximum dimension of features. Each feature represents an axis.
- #define MAX_FEATURE_SAMPLES 200

  Maximum number of feature samples that is allowed to train a model.
- #define MAX_FEATURE_INSTANCES 10

  Maximum number of feature instances. Each feature instance is linked to model instances.
- #define MAX_MODEL_INSTANCES MAX_GMM + MAX_SVM

  Maximum number of model instances. The total number is the sum of the number GMMs and SVMs.
- #define **MAX_SUBMODELS** MAX_MODEL_INSTANCES
- #define DEBUG_ENABLE_TX_VARIABLE_IS_ON 0

  0: not debug (not using the modelInstance->enableTransmit variable), 1: debugging (it uses it modelInstance->enableTransmit).

Typedefs

- typedef struct FeatureInstance FeatureInstance

  Feature Instance structure contains the buffers for feature samples and other information.
- typedef struct FeatureCollection FeatureCollection

  Feature Collection structure contains the poinsters of FeatureInstance structures.
- typedef struct GaussComponent GaussComponent

  A Gaussian component structure in the mixture model.
- typedef struct GmmModel GmmModel

  A Gaussian mixture model (GMM) structure.
- typedef struct svm_node_embedded svm_node_embedded

  A structure for each support vector.
- typedef struct OcsvmModel OcsvmModel

  A One Class Support Vector Machine (OC-SVM) structure.
- typedef struct ModelInstance ModelInstance

  Model Instance Structure contains the buffers for feature samples and other information.
- typedef struct ModelCollection ModelCollection

  Model Collection structure contains the poinsters of ModelInstance structures.

Functions

- void initFeatureCollection (struct FeatureCollection ∗featureCollection, struct FeatureInstance ∗feature←Instance)

  Initializes the FeatureCollection structure.
- void initModelCollection (struct ModelCollection ∗models, struct ModelInstance ∗modelInstance, union Model ∗ml_model)

  Initializes the ModelCollection structure.
- void addToFeatureBuffers (struct Globals ∗gbls)

  Adds the real-time features to the buffer in FeatureInstance.
- bool initGMM (struct ModelInstance ∗modelInstance)

  Initializes GMM parameters.

- void runGMMEM (struct ModelInstance ∗modelInstance)

    Executes expectation-maximization (EM) algorithm to estimate GMM parameters.
- void trainGMM (struct ModelCollection ∗models, uint8_t modelID)

    Executes TRAINING of GMM instance.
- float **getLastFeatureValueAdded** (FeatureInstance ∗featureInstance)
- int8_t **idxViaFindOrAssignModelID** (ModelCollection ∗models, uint8_t modelID)
- void disableAllModels (struct ModelCollection ∗models)

    Stops all the active models.
- bool enableRunModel (struct ModelCollection ∗modelCollection, uint8_t modelID)

    Enbales a model with model ID to run.
- bool enableModel (struct ModelCollection ∗modelCollection, uint8_t modelID)

    Enables model instances.
- void incrementFeatureBufferIndices (struct Globals ∗gbls)

    increases the featuer buffer indices.
- void detectAnomalyInGMM (struct ModelInstance ∗modelInstance)

    Anomaly detection function for GMM.
- void computeZscore (float ∗xn, float x, float mu, float sigma)

    Computes zscore.
- void incrementFeatureBufferIndicesModels (struct Globals ∗gbls)

    Increment Feature Buffer Indices of Models.
- void incrementFeatureBufferIndicesStartEnd (struct ModelInstance ∗model)

    Increases feature buffer indices for a model instance.
- bool checkReversed (int startPosition, int endPosition)

    Checks whether a moving window is reversed or not.
- void inputPositions (struct ModelInstance ∗modelInstance, int ∗startPos, int ∗endPos)

    Finds a window position.
- void zscoreFeatBuffer (struct FeatureInstance ∗featureInstance)

    Computes zscore (standardized by the given std and mean).
- void refreshGMM (struct ModelInstance ∗modelInstance)

    Refreshes GMM parameters.
- bool **checkMixProb** (struct GmmModel ∗gmm_model)
- bool **computeR** (struct ModelInstance ∗modelInstance, float R[ ][2])
- void normalizeFeatures (struct ModelInstance ∗modelInstance)

    Normalizes the features within a moving window.
- void normalizeFeatWindow (struct FeatureInstance ∗feat_inst, int sPos, int ePos, bool reversed)
- float **sumBuffer** (float ∗pBuffer, int startPos, int endPos, bool reversed)
- float **sumSquaredBuffer** (float ∗pBuffer1, float ∗pBuffer2, int startPos, int endPos, bool reversed)
- void **trainGMM_instance** (struct ModelInstance ∗modelInstance)
- void **clearFeatureBuffers_Instance** (struct ModelInstance ∗modelInstance)
- void trainOCSVM (struct ModelCollection ∗modelCollection, uint8_t modelID)

    Executes TRAINING of SVM instance.
- void trainOCSVM_malloc (struct ModelInstance ∗modelInstance)

    Executes OC-SVM algorithm to estimate SVs.
- void parse_svm_param (struct OcsvmModel ∗ocsvm, struct svm_parameter ∗param)

    To parse the input parameters for SVM before training.
- void save_model_instance (struct svm_model ∗model, struct OcsvmModel ∗ocsvm)

    To save the trained SVM model into a ModelInstance structure.
- void detectAnomalyInSVM (struct ModelInstance ∗modelInstance)

    Anomaly detection function for SVM.
- void **trainOCSVM_instance** (struct ModelInstance ∗modelInstance)
- void detectAnomalyInEnsemble (struct ModelInstance ∗modelInstance)

Anomaly detection function for Ensemble Models.

- void **trainEnsemble** (struct ModelCollection ∗modelCollection)
- void addToFeatBuffer (struct Globals ∗gbls, struct FeatureInstance ∗feature_instance)

    Feature Operation Functions control functions for machine learning feature operation for creating and saving features from raw sensor data.

- bool **get_feature_attributes** (struct FeatureCollection ∗features, uint8_t idx, sensor_t ∗sensor, axis_t ∗axis, feature_t ∗feature)
- bool **delete_feature_by_attributes** (struct FeatureCollection ∗features, sensor_t sensor, axis_t axis, feature_t feature)
- bool **delete_feature_by_index** (struct FeatureCollection ∗features, uint8_t idx)
- bool add_feature (uint8_t ∗idx, struct FeatureCollection ∗features, sensor_t sensor, axis_t axis, feature_t feature)

    Adds features by (sensor_t, axis_t, feature_t).

- bool get_feature_number (uint8_t ∗idx, struct FeatureCollection ∗features, sensor_t sensor, axis_t axis, feature_t feature)

    Gets feature instance number.

- bool get_model_idx_instance (uint8_t ∗idx, bool ∗is_enabled, struct ModelCollection ∗models, uint8_↩ t modelID)

    Model Operation Functions control functions for machine learning models operation such as training, testing, etc.

- bool trainModel (struct ModelCollection ∗modelCollection, uint8_t modelID)

    Executes TRAINING of model instances.

- bool runModel (struct ModelInstance ∗modelInstance, bool ∗passFail, float ∗likelihood)

    Execute RUN mode of a model instance.

5.32.1 Detailed Description

Data structures and function prototypes for the machine learning library.

This contains the data structure and function proto types for the machine learning algorithm library. General ones are described here. Additional structures and prototypes for GMM and SVM are in "gmm_utility.h" and "svm.h" respectively.

5.32.2 Typedef Documentation

5.32.2.1 FeatureCollection

```
typedef struct FeatureCollection FeatureCollection
```

Feature Collection structure contains the poinsters of FeatureInstance structures.

Potentially, control parameters for a collection of Feature Instances may be added here.

5.32.2.2 FeatureInstance

```
typedef struct FeatureInstance FeatureInstance
```

Feature Instance structure contains the buffers for feature samples and other information.

Feature type, sensor type, and sensor axis are used to identify feature instance. fBufferNormalized is the buffer for normalized features (zscore) from fBuffer, fBufferMean, and fBufferStd. The other variables in FeatureInstance structure are control variables.

5.32.2.3 GaussComponent

`typedef struct GaussComponent GaussComponent`

A Gaussian component structure in the mixture model.

This structure contains the computational information as well as the parameter set of a single Gaussian component in the mixture model. GaussCompoentn is included in GmmModel structure.

5.32.2.4 GmmModel

`typedef struct GmmModel GmmModel`

A Gaussian mixture model (GMM) structure.

GMM parameters as well as trained GMM by expectation-maximization (EM) algorithm. These are linked to Model←Instance.

5.32.2.5 ModelCollection

`typedef struct ModelCollection ModelCollection`

Model Collection structure contains the poinsters of ModelInstance structures.

Potentially, control parameters for a collection of Model Instances may be added more.

5.32.2.6 ModelInstance

`typedef struct ModelInstance ModelInstance`

Model Instance Structure contains the buffers for feature samples and other information.

Model type and ID are used to identify model instance. The selected feature instances are linked to a model instance. GMM and OC-SVM are linked to a model instance as a union. So, only one model is used in a model instance.

5.32.2.7 OcsvmModel

`typedef struct OcsvmModel OcsvmModel`

A One Class Support Vector Machine (OC-SVM) structure.

OC-SVM parameters as well as trained model by use of LIBSVM. This structure is linked to ModelInstance.

5.32.2.8 svm_node_embedded

`typedef struct svm_node_embedded svm_node_embedded`

A structure for each support vector.

This structure was defined for embedded systems with reduced memory usage, rather than using 8 bytes double type. It follows the same structure of LIBSVM.

5.32.3 Function Documentation

5.32.3.1 add_feature()

```
bool add_feature (
            uint8_t * idx,
            struct FeatureCollection * featureCollection,
            sensor_t sensor,
            axis_t axis,
            feature_t feature )
```

Adds features by (sensor_t, axis_t, feature_t).

This function is to add and enable feature instance with coordination (sensor_t, axis_t, feature_t).

Parameters

idx	The output that indicates the resulting index of featureInstance.
featureCollection	The pointer of FeatureCollection structure.
sensor	Sensor type information.
axis	Sensor axis information.
feature	Feature type information.

Returns

true / false.

Definition at line 526 of file machineLearning_subsystem.c.

5.32.3.2 addToFeatBuffer()

```
void addToFeatBuffer (
            struct Globals * gbls,
            struct FeatureInstance * featureInstance )
```

Feature Operation Functions control functions for machine learning feature operation for creating and saving features from raw sensor data.

Feature Operation Functions control functions for machine learning feature operation for creating and saving features from raw sensor data.

This function adds a feature sample to the feature buffer in feature instance. The Globals structure contains four different types sensors where each feature instance takes feature samples from only one of the sensors. Definitions - F_USING_ACCEL, F_USING_MAG, F_USING_GYRO, and F_1DOF_P_BASIC separate the sensor types.

Parameters

gbls	The pointer of Globals structure.
featureInstance	The pointer of FeatureInstance structure.

Returns

void

Definition at line 196 of file machineLearning_subsystem.c.

5.32.3.3 addToFeatureBuffers()

```
void addToFeatureBuffers (
            struct Globals * gbls )
```

Adds the real-time features to the buffer in FeatureInstance.

This function is to add the current features computed in real time into the buffer of FeatureInstance. It first finds which feature instance is enabled, and then calls addToFeatBuffer() function in order to put the values into the enabled feature instance.

Parameters

gbls	The pointer of Globals structure.

Returns

void.

Definition at line 169 of file machineLearning_subsystem.c.

5.32.3.4 checkReversed()

```
bool checkReversed (
            int startPosition,
            int endPosition )
```

Checks whether a moving window is reversed or not.

This function is to check whether a moving window is reversed or not. If the window is reversed, then the start↩ Position will be bigger than endPosition. If not, startPosition will be smaller than endPosition.

Parameters

startPosition	The index that indicates the start position of a moving window.
endPosition	The index that indicates the end position of a moving window.

Returns

true / false.

Definition at line 342 of file machineLearning_subsystem.c.

5.32.3.5 computeZscore()

```
void computeZscore (
            float * xn,
            float x,
            float mu,
            float sigma )
```

Computes zscore.

This function computes zscore instantly.

Parameters

xn	The pointer for the output of this function.
x	The feature to be standardized.
mu	Mean.
sigma	Standard deviation.

Returns

void.

Definition at line 254 of file machineLearning_subsystem.c.

5.32.3.6 detectAnomalyInEnsemble()

```
void detectAnomalyInEnsemble (
            struct ModelInstance * modelInstance )
```

Anomaly detection function for Ensemble Models.

This function is for anomaly detection using ensemble models. Soft and hard decisions are made.

Parameters

modelInstance	The pointer of ModelInstance structure.

Returns

void.

Definition at line 743 of file machineLearning_subsystem.c.

5.32.3.7 detectAnomalyInGMM()

```
void detectAnomalyInGMM (
            struct ModelInstance * modelInstance )
```

Anomaly detection function for GMM.

This function is for anomaly detection using GMM. Soft and hard decisions are made.

Parameters

modelInstance	The pointer of ModelInstance structure.

Returns

void.

Definition at line 669 of file machineLearning_subsystem.c.

5.32.3.8 detectAnomalyInSVM()

```
void detectAnomalyInSVM (
            struct ModelInstance * modelInstance )
```

Anomaly detection function for SVM.

This function is for anomaly detection using OC-SVM. Soft and hard decisions are made.

Parameters

modelInstance	The pointer of ModelInstance structure.

Returns

void.

Definition at line 699 of file machineLearning_subsystem.c.

5.32.3.9 disableAllModels()

```
void disableAllModels (
            struct ModelCollection * models )
```

Stops all the active models.

This function stops the active models by setting isEnabled to false at each model instance.

Parameters

modelCollection	The pointer of ModelCollection structure.

Returns

void.

Definition at line 136 of file machineLearning_subsystem.c.

5.32.3.10 enableModel()

```
bool enableModel (
            struct ModelCollection * modelCollection,
            uint8_t modelID )
```

Enables model instances.

This function enables modelInstance with model ID.

Parameters

modelCollection	The pointer of ModelCollection structure.
modelID	Model ID.

Returns

true / false True if success.

Definition at line 608 of file machineLearning_subsystem.c.

5.32.3.11 enableRunModel()

```
bool enableRunModel (
            struct ModelCollection * modelCollection,
            uint8_t modelID )
```

Enbales a model with model ID to run.

This function enables a model that is specified by model ID to run. iStage of modelInstance is set to RUN_COM↩ MAND, if the model instance has been trained.

Parameters

modelCollection	The pointer of ModelCollection structure.
modelID	Model ID.

Returns

true / false True if success.

Definition at line 587 of file machineLearning_subsystem.c.

5.32.3.12 get_feature_number()

```
bool get_feature_number (
            uint8_t * idx,
            struct FeatureCollection * featureCollection,
            sensor_t sensor,
            axis_t axis,
            feature_t feature )
```

Gets feature instance number.

This function gets the index of FeatureInstance structure from the coordination of sensor_type, sensor_axis, and feature_type.

Parameters

idx	The output that indicates the resulting index of featureInstance.
featureCollection	The pointer of FeatureCollection structure.
sensor	Sensor type information.
axis	Sensor axis information.
feature	Feature type information.

Returns

true / false.

Definition at line 499 of file machineLearning_subsystem.c.

5.32.3.13 get_model_idx_instance()

```
bool get_model_idx_instance (
            uint8_t * idx,
            bool * isEnabled,
            struct ModelCollection * modelCollection,
            uint8_t modelID )
```

Model Operation Functions control functions for machine learning models operation such as training, testing, etc.

Model Operation Functions control functions for machine learning models operation such as training, testing, etc.

This function searches model IDs of modelInstance structures and assigns the ID number and isEnabled.

Parameters

idx	The output that indicates modelInstance index.
isEnabled	The output that indicates whether the modelInstance is enabled or not.
modelCollection	The pointer of ModelCollection structure
modelID	Model ID.

Returns

true / false True if success.

Definition at line 476 of file machineLearning_subsystem.c.

5.32.3.14 incrementFeatureBufferIndices()

```
void incrementFeatureBufferIndices (
            struct Globals * gbls )
```

increases the featuer buffer indices.

This function increases the feature buffer indices, if the feature instance is enabled. When the index exceeds M↩AX_FEATURE_SAMPLES which equivalently defines the maximum size of buffer, the index is reset and creases again. iBufferExceeded indicates whether this occured or not.

Parameters

gbls	The pointer of Globals structure.

Returns

void

Definition at line 268 of file machineLearning_subsystem.c.

5.32.3.15 incrementFeatureBufferIndicesModels()

```
void incrementFeatureBufferIndicesModels (
            struct Globals * gbls )
```

Increment Feature Buffer Indices of Models.

This function increase the indices of feature buffers, if the model instance is enabled.

Parameters

gbls	The pointer of Globals structure.

Returns

void.

Definition at line 293 of file machineLearning_subsystem.c.

5.32.3.16 incrementFeatureBufferIndicesStartEnd()

```
void incrementFeatureBufferIndicesStartEnd (
            struct ModelInstance * modelInstance )
```

Increases feature buffer indices for a model instance.

This function increases feature buffer indices for the given model instance. It checks whether the indices are within a moving window that is reversed or not.

Parameters

modelInstance	The pointer of ModelInstance structure.

Returns

void.

Definition at line 311 of file machineLearning_subsystem.c.

5.32.3.17 initFeatureCollection()

```
void initFeatureCollection (
            struct FeatureCollection * featureCollection,
            struct FeatureInstance * featureInstance )
```

Initializes the FeatureCollection structure.

The FeatureCollection structure contains pointers of feature instances. This function initializes the address of feature instances created, sensor type, sensor axis, feature type, buffer counter, and indicators such as iBufferExceeded, isEnabled, and isNormalized.

Parameters

features	The pointer of FeatureCollection structure.
featureInstance	The first pointer of FeatureInstance array structure (if multiple feature instances are defined).

Returns

void.

Definition at line 63 of file machineLearning_subsystem.c.

5.32.3.18 initGMM()

```
bool initGMM (
            struct ModelInstance * modelInstance )
```

Initializes GMM parameters.

This function initializes GMM parameters. At every training time, this function is called to refresh the GMM parameters. If opt_num_components is less than init_maxGaussComponents, GMM increases its own components and then refresh the parameters. This way helps to smoothly adapt GMM over time.

Parameters

modelInstance	The pointer of ModelInstance structure.

Returns

true / false True if success.

Definition at line 925 of file machineLearning_subsystem.c.

5.32.3.19 initModelCollection()

```
void initModelCollection (
            struct ModelCollection * modelCollection,
            struct ModelInstance * modelInstance,
            union Model * model )
```

Initializes the ModelCollection structure.

The ModelCollection is initialized with the number of models that are going to process, training rate, and model↵ Instance structures. The ModelInstance structure contains pointers of feature instances. This function initializes the address of model instances created, feature_dimension, model ID, decision results (soft / hard), and indicators such as isEnabled and isTrained. Each modelInstance also contains startPositionBuffer and endPositionBuffer that are used for a moving window within a feature buffer. That moving window is to pick a given number of latest features from the buffer in real time. The ModelInstance structure also contains the pointers of feature instances and union of model types (GMM / SVM). Each modelInstance takes only one of GMM and SVM models.

Parameters

modelCollection	The pointer of ModelCollection structure.
modelInstance	The pointer of ModelInstance structure.
model	The pointer of union Model that contains struct GmmModel and struct OcsvmModel.

Returns

void.

Definition at line 101 of file machineLearning_subsystem.c.

5.32.3.20 inputPositions()

```
void inputPositions (
            struct ModelInstance * modelInstance,
            int * startPos,
            int * endPos )
```

Finds a window position.

This function finds the start and end positions of a moving window.

Parameters

modelInstance	The pointer of ModelInstance structure.
startPos	The output that contains the start position of window.
endPos	The output that contains the end position of window.

Returns

void.

Definition at line 357 of file machineLearning_subsystem.c.

5.32.3.21 normalizeFeatures()

```
void normalizeFeatures (
            struct ModelInstance * modelInstance )
```

Normalizes the features within a moving window.

This function is to normalize the features within a moving window whose start and end positions are maintained in ModelInstance structure. The function contains a subroutine for checking whether those positions are reversed or not. normalizedFeatWindow() is referred to.

Parameters

modelInstance	The pointer of ModelInstance structure.

Returns

void.

Definition at line 388 of file machineLearning_subsystem.c.

5.32.3.22 normalizeFeatWindow()

```
void normalizeFeatWindow (
            struct FeatureInstance * featureInstance,
            int sPos,
            int ePos,
            bool reversed )
```

Normalizes features within a moving window.

```
This function is to normalize the features within a moving window. The start and end poistions of the window a
and stored in FeatureInstance structure.
isNormalized is set to true at the end of this function.

@param  featureInstance The pointer of FeatureInstance structure.
@param  sPos    The start position of moving window.
@param  ePos    The end position of moving window.
@param  reversed        The indicator whether the moving window is reversed or not.
@return void.
```

Definition at line 421 of file machineLearning_subsystem.c.

5.32.3.23 parse_svm_param()

```
void parse_svm_param (
            struct OcsvmModel * ocsvm,
            struct svm_parameter * param )
```

To parse the input parameters for SVM before training.

This function is to parse the input parameters for SVM before training.

Parameters

ocsvm	The pointer of struct OcsvmModel which is a union of Model in ModelInstance structure.
param	The pointer of struct svm_parameter which is defined in LIBSVM.

Returns

void.

Definition at line 1210 of file machineLearning_subsystem.c.

5.32.3.24 refreshGMM()

```
void refreshGMM (
            struct ModelInstance * modelInstance )
```

Refreshes GMM parameters.

This function refreshes GMM parameters. Somtimes, GMM parameters are ill conditioned. This function is called to refresh the GMM parameters. If opt_num_components is less than init_maxGaussComponents, GMM increases its own components and then refresh the parameters. This way helps to smoothly adapt GMM over time.

Parameters

modelInstance	The pointer of ModelInstance structure.

Returns

true / false True if success.

Definition at line 1036 of file machineLearning_subsystem.c.

5.32.3.25 runGMMEM()

```
void runGMMEM (
            struct ModelInstance * modelInstance )
```

Executes expectation-maximization (EM) algorithm to estimate GMM parameters.

This function executes EM algorithm to estimate GMM parameters. With an initial set of parameters, it first computes minimum description length (MDL) criterion, which is used to find the optimal number of Gaussian components, by computeMDL_GMM(). Reducing the number of mixture components upto one, it evaluates MDL for different number of mixture components. When it reduces the number of components, reduceModelOrder() sums two adjacent componets and discard one of the two. The adjacency is measured by mean vectors of every given pairs of components. For every case of mixture components, EM algoritm is processed by calling reestimate() and computeLL_regroup() in computeMDL_GMM().

Parameters

modelInstance	The pointer of ModelInstance structure.

Returns

void.

Definition at line 984 of file machineLearning_subsystem.c.

5.32.3.26 runModel()

```
bool runModel (
            struct ModelInstance * modelInstance,
            bool * passFail,
            float * likelihood )
```

Execute RUN mode of a model instance.

This function executes the run mode of a model instance. Depending on the model type (GMM / SVM), given the trained model, detectionAnomalyInGMM(), detectionAnomalyInSVM(), or detectionAnomalyInEnsemble() is called.

Parameters

modelInstance	The pointer of ModelInstance structure.
passFail	Hard decision restult. Pass: positive (normal). Fail: negative (anomaly).
likelihood	Soft decition restult. The output of density function built by GMM or SVM.

Returns

true / false True if success.

Definition at line 628 of file machineLearning_subsystem.c.

5.32.3.27 save_model_instance()

```
void save_model_instance (
            struct svm_model * model,
            struct OcsvmModel * ocsvm )
```

To save the trained SVM model into a ModelInstance structure.

This function is to save the trained SVM model by LIBSVM into a ModelInstance structure. Type conversion is done.

Parameters

model	The pointer of struct svm_model.
ocsvm	The pointer of struct OcsvmModel which is a union of Model in ModelInstance structure.

Returns

void.

Definition at line 1179 of file machineLearning_subsystem.c.

5.32.3.28 trainGMM()

```
void trainGMM (
            struct ModelCollection * modelCollection,
            uint8_t modelID )
```

Executes TRAINING of GMM instance.

This function is for training a GMM instance. If the model instance is enabled and the features are normalized, then Expectation-Maximization (EM) algorithm runs. initGMM() is called to control GMM parameters before EM is processed. runGMMEM() executes the EM algorithm. Before calling those functions, it is assumed to check whether the moving window of feature buffer is reversed or not. Once TRAINING is done, isTrained is set to true.

Parameters

modelCollection	The pointer of ModelCollection structure.
modelID	Model ID.

Returns

true / false True if success.

Definition at line 823 of file machineLearning_subsystem.c.

5.32.3.29 trainModel()

```
bool trainModel (
            struct ModelCollection * modelCollection,
            uint8_t modelID )
```

Executes TRAINING of model instances.

This function is for training a model instance. Each modelInstance contains a model type (GMM or SVM or Ensemble), and calls one of the training functions (trainGMM(), trainOCSVM(), and trainALL()). Before calling those functions, it is assumed to normalize features because normalization can help models to avoid undesired situations (e.g., sigularity of covariance matrix in GMM). Also, normalization scales down arbitrary features down to standardized ones by std and mean.

Parameters

modelCollection	The pointer of ModelCollection structure.
modelID	Model ID.

Returns

true / false True if success.

Definition at line 757 of file machineLearning_subsystem.c.

5.32.3.30 trainOCSVM()

```
void trainOCSVM (
            struct ModelCollection * modelCollection,
            uint8_t modelID )
```

Executes TRAINING of SVM instance.

This function is for training a SVM instance. If the model instance is enabled and the features are normalized, then one class support vector machine (OC-SVM) algorithm runs. Before calling this function, it is assumed to check whether the moving window of feature buffer is reversed or not. Once TRAINING is done, isTrained is set to true.

Parameters

modelCollection	The pointer of ModelCollection structure.
modelID	Model ID.

Returns

true / false True if success.

Definition at line 1103 of file machineLearning_subsystem.c.

5.32.3.31 trainOCSVM_malloc()

```
void trainOCSVM_malloc (
            struct ModelInstance * modelInstance )
```

Executes OC-SVM algorithm to estimate SVs.

This function is for execution of OC-SVM. As a result of the algorithm, a set of support vectors (SVs) are provided, which build SVM model. The parameter nu controls the bounds on the number of SVs and fraction of anomaly in training data set. LIBSVM library is utilized to build the SVM model. Since LIBSVM uses dynamic memory allocation, this function follows the methodology by using the capability provided by FreeRTOS. The dynamic memory allocation here is based on heap_4.c. For the embedded system we are working on, we limited the maximum size of feature buffers, which means the maximum number of SVs is also limited. This may reduce the risk of use of dynamic memory allocation in the current implementation.

Parameters

modelInstance	The pointer of ModelInstance structure.

Returns

void.

Definition at line 1132 of file machineLearning_subsystem.c.

5.32.3.32 zscoreFeatBuffer()

```
void zscoreFeatBuffer (
            struct FeatureInstance * featureInstance )
```

Computes zscore (standardized by the given std and mean).

This function normalizes the feature sample in fBuffer and save it in fBufferNormalized. iBufferCount is the current index of the feature.

Parameters

featureInstance	The pointer of FeatureInstance structure.

Returns

void

Definition at line 237 of file machineLearning_subsystem.c.

5.33 magnetic.c File Reference

Lower level magnetic calibration interface.

```
#include "anomaly_detection.h"
#include "math.h"
#include "stdlib.h"
#include "time.h"
```

5.33.1 Detailed Description

Lower level magnetic calibration interface.

Many developers can utilize the NXP Sensor Fusion Library without ever making any adjustment to the lower level magnetic calibration functions defined in this file.

5.34 magnetic.h File Reference

Lower level magnetic calibration interface.

Data Structures

- struct MagBuffer
- struct MagCalibration

    Magnetic Calibration Structure.

Macros

- #define **F_USING_MAG** 0x0002

Magnetic Calibration Constants

- #define MAGBUFFSIZEX 14
  x dimension in magnetometer buffer (12x24 equals 288 elements)
- #define MAGBUFFSIZEY (2 ∗ MAGBUFFSIZEX)
  y dimension in magnetometer buffer (12x24 equals 288 elements)
- #define MINMEASUREMENTS4CAL 110
  minimum number of measurements for 4 element calibration
- #define MINMEASUREMENTS7CAL 220
  minimum number of measurements for 7 element calibration
- #define MINMEASUREMENTS10CAL 330
  minimum number of measurements for 10 element calibration
- #define MAXMEASUREMENTS 360
  maximum number of measurements used for calibration
- #define CAL_INTERVAL_SECS 300
  300s or 5min interval for regular calibration checks
- #define MINBFITUT 10.0F
  minimum acceptable geomagnetic field B (uT) for valid calibration
- #define MAXBFITUT 90.0F
  maximum acceptable geomagnetic field B (uT) for valid calibration
- #define FITERRORAGINGSECS 86400.0F
  24 hours: time (s) for fit error to increase (age) by e=2.718
- #define MESHDELTACOUNTS 50
  magnetic buffer mesh spacing in counts (here 5uT)
- #define DEFAULTB 50.0F
  default geomagnetic field (uT)

Functions

Function prototypes for functions in magnetic.c

These functions comprise the core of the magnetic calibration features of the library. Parameter descriptions are not included here, as details are provided in anomaly_detection.h.

- void **fInitializeMagCalibration** (struct MagCalibration ∗pthisMagCal, struct MagBuffer ∗pthisMagBuffer)
- void **iUpdateMagBuffer** (struct MagBuffer ∗pthisMagBuffer, struct MagSensor ∗pthisMag, int32_t loopcounter)
- void **fInvertMagCal** (struct MagSensor ∗pthisMag, struct MagCalibration ∗pthisMagCal)
- void **fRunMagCalibration** (struct MagCalibration ∗pthisMagCal, struct MagBuffer ∗pthisMagBuffer, struct MagSensor ∗pthisMag, int32_t loopcounter)
- void **fUpdateMagCalibration4** (struct MagCalibration ∗pthisMagCal, struct MagBuffer ∗pthisMagBuffer, struct MagSensor ∗pthisMag)
- void **fUpdateMagCalibration7** (struct MagCalibration ∗pthisMagCal, struct MagBuffer ∗pthisMagBuffer, struct MagSensor ∗pthisMag)
- void **fUpdateMagCalibration10** (struct MagCalibration ∗pthisMagCal, struct MagBuffer ∗pthisMagBuffer, struct MagSensor ∗pthisMag)
- void **fUpdateMagCalibration4Slice** (struct MagCalibration ∗pthisMagCal, struct MagBuffer ∗pthisMagBuffer, struct MagSensor ∗pthisMag)
- void **fUpdateMagCalibration7Slice** (struct MagCalibration ∗pthisMagCal, struct MagBuffer ∗pthisMagBuffer, struct MagSensor ∗pthisMag)
- void **fUpdateMagCalibration10Slice** (struct MagCalibration ∗pthisMagCal, struct MagBuffer ∗pthisMagBuffer, struct MagSensor ∗pthisMag)

5.34.1 Detailed Description

Lower level magnetic calibration interface.

Many developers can utilize the NXP Sensor Fusion Library without ever making any adjustment to the lower level magnetic calibration functions defined in this file.

5.35 matrix.c File Reference

Matrix manipulation functions.

```
#include "stdio.h"
#include "math.h"
#include "stdlib.h"
#include "time.h"
#include "anomaly_detection.h"
#include "matrix.h"
```

Macros

- #define **CORRUPTMATRIX** 0.001F
- #define **NITERATIONS** 15
- #define **NITERATIONS** 15

Functions

- void f3x3matrixAeqI (float A[ ][3])

    function sets the 3x3 matrix A to the identity matrix
- void f3x3matrixAeqB (float A[ ][3], float B[ ][3])

    function sets 3x3 matrix A to 3x3 matrix B
- void fmatrixAeqI (float ∗A[ ], int16 rc)

    function sets the matrix A to the identity matrix
- void f3x3matrixAeqScalar (float A[ ][3], float Scalar)

    function sets every entry in the 3x3 matrix A to a constant scalar
- void f3x3matrixAeqAxScalar (float A[ ][3], float Scalar)

    function multiplies all elements of 3x3 matrix A by the specified scalar
- void f3x3matrixAeqMinusA (float A[ ][3])

    function negates all elements of 3x3 matrix A
- void f3x3matrixAeqInvSymB (float A[ ][3], float B[ ][3])
- float f3x3matrixDetA (float A[ ][3])

    function calculates the determinant of a 3x3 matrix
- float f2x2matrixDetA (float A[ ][2])

    function calculates the determinant of a 2x2 matrix
- void fEigenCompute10 (float A[ ][10], float eigval[ ], float eigvec[ ][10], int8 n)
- void fEigenCompute4 (float A[ ][4], float eigval[ ], float eigvec[ ][4], int8 n)
- void **fComputeEigSlice** (float fmatA[10][10], float fmatB[10][10], float fvecA[10], int8 i, int8 j, int8 iMatrixSize)
- void fmatrixAeqInvA (float ∗A[ ], int8 iColInd[ ], int8 iRowInd[ ], int8 iPivot[ ], int8 isize, int8 ∗pierror)
- void fveqRu (float fv[ ], float fR[ ][3], float fu[ ], int8 itranspose)
- void fVeq3x3AxV (float V[3], float A[ ][3])

    function multiplies the 3x1 vector V by a 3x3 matrix A

5.35.1 Detailed Description

Matrix manipulation functions.

Contains functions for basic manipulation of 3x3 matrices

5.35.2 Function Documentation

5.35.2.1 f3x3matrixAeqInvSymB()

```
void f3x3matrixAeqInvSymB (
            float A[ ][3],
            float B[ ][3] )
```

function directly calculates the symmetric inverse of a symmetric 3x3 matrix only the on and above diagonal terms in B are used and need to be specified

Definition at line 172 of file matrix.c.

5.35.2.2 fEigenCompute10()

```
void fEigenCompute10 (
            float A[ ][10],
            float eigval[ ],
            float eigvec[ ][10],
            int8 n )
```

function computes all eigenvalues and eigenvectors of a real symmetric matrix A[0..n-1][0..n-1] stored in the top left of a 10x10 array A[10][10]

Parameters

A	real symmetric matrix A[0..n-1][0..n-1]
eigval	eigval[0..n-1] returns the eigenvalues of A[][].
eigvec	eigvec[0..n-1][0..n-1] returns the normalized eigenvectors of A[][]
n	n can vary up to and including 10 but the matrices A and eigvec must have 10 columns.

Definition at line 244 of file matrix.c.

5.35.2.3 fEigenCompute4()

```
void fEigenCompute4 (
            float A[ ][4],
```

```
float eigval[],
float eigvec[][4],
int8 n )
```

function computes all eigenvalues and eigenvectors of a real symmetric matrix A[0..n-1][0..n-1] stored in the top left of a 4x4 array A[4][4] A[][] is changed on output. The eigenvectors are not sorted by value. This function is identical to eigencompute10 except for the workaround for 4x4 matrices since C cannot handle functions accepting matrices with variable numbers of columns.

Parameters

eigval	eigval[0..n-1] returns the eigenvalues of A[][].
eigvec	eigvec[0..n-1][0..n-1] returns the normalized eigenvectors of A[][]
n	n can vary up to and including 4 but the matrices A and eigvec must have 4 columns.

Definition at line 417 of file matrix.c.

5.35.2.4 fmatrixAeqI()

```
void fmatrixAeqI (
            float * A[],
            int16 rc )
```

function sets the matrix A to the identity matrix

Parameters

A	pointer to the matrix
rc	dimension of the matrix

Definition at line 91 of file matrix.c.

5.35.2.5 fmatrixAeqInvA()

```
void fmatrixAeqInvA (
            float * A[],
            int8 iColInd[],
            int8 iRowInd[],
            int8 iPivot[],
            int8 isize,
            int8 * pierror )
```

function uses Gauss-Jordan elimination to compute the inverse of matrix A in situ on exit, A is replaced with its inverse

Definition at line 676 of file matrix.c.

5.35.2.6 fVeq3x3AxV()

```
void fVeq3x3AxV (
            float V[3],
            float A[ ][3] )
```

function multiplies the 3x1 vector V by a 3x3 matrix A

Parameters

V	used for both input and output

Definition at line 881 of file matrix.c.

5.35.2.7 fveqRu()

```
void fveqRu (
            float fv[ ],
            float fR[ ][3],
            float fu[ ],
            int8 itranspose )
```

function rotates 3x1 vector u onto 3x1 vector using 3x3 rotation matrix fR. the rotation is applied in the inverse direction if itranpose is true

Parameters

fv	3x1 output vector
fR	rotation matrix
fu	3x1 input vector
itranspose	true if inverse direction desired

Definition at line 830 of file matrix.c.

5.36 matrix.h File Reference

Matrix manipulation functions.

Functions

- void f3x3matrixAeqI (float A[ ][3])

  function sets the 3x3 matrix A to the identity matrix
- void f3x3matrixAeqB (float A[ ][3], float B[ ][3])

  function sets 3x3 matrix A to 3x3 matrix B
- void fmatrixAeqI (float ∗A[ ], int16 rc)

function sets the matrix A to the identity matrix
- void f3x3matrixAeqScalar (float A[ ][3], float Scalar)

  function sets every entry in the 3x3 matrix A to a constant scalar
- void f3x3matrixAeqInvSymB (float A[ ][3], float B[ ][3])
- void f3x3matrixAeqAxScalar (float A[ ][3], float Scalar)

  function multiplies all elements of 3x3 matrix A by the specified scalar
- void f3x3matrixAeqMinusA (float A[ ][3])

  function negates all elements of 3x3 matrix A
- float f3x3matrixDetA (float A[ ][3])

  function calculates the determinant of a 3x3 matrix
- float f2x2matrixDetA (float A[ ][2])

  function calculates the determinant of a 2x2 matrix
- void fEigenCompute10 (float A[ ][10], float eigval[ ], float eigvec[ ][10], int8 n)
- void fEigenCompute4 (float A[ ][4], float eigval[ ], float eigvec[ ][4], int8 n)
- void **fComputeEigSlice** (float fmatA[10][10], float fmatB[10][10], float fvecA[10], int8 i, int8 j, int8 iMatrixSize)
- void fmatrixAeqInvA (float ∗A[ ], int8 iColInd[ ], int8 iRowInd[ ], int8 iPivot[ ], int8 isize, int8 ∗pierror)
- void fveqRu (float fv[ ], float fR[ ][3], float fu[ ], int8 itranspose)
- void fVeq3x3AxV (float V[3], float A[ ][3])

  function multiplies the 3x1 vector V by a 3x3 matrix A

5.36.1 Detailed Description

Matrix manipulation functions.

Contains functions for basic manipulation of 3x3 matrices

5.36.2 Function Documentation

5.36.2.1 f3x3matrixAeqInvSymB()

```
void f3x3matrixAeqInvSymB (
            float A[ ][3],
            float B[ ][3] )
```

function directly calculates the symmetric inverse of a symmetric 3x3 matrix only the on and above diagonal terms in B are used and need to be specified

Definition at line 172 of file matrix.c.

5.36.2.2 fEigenCompute10()

```
void fEigenCompute10 (
            float A[ ][10],
            float eigval[ ],
            float eigvec[ ][10],
            int8 n )
```

function computes all eigenvalues and eigenvectors of a real symmetric matrix A[0..n-1][0..n-1] stored in the top left of a 10x10 array A[10][10]

Parameters

A	real symmetric matrix A[0..n-1][0..n-1]
eigval	eigval[0..n-1] returns the eigenvalues of A[][].
eigvec	eigvec[0..n-1][0..n-1] returns the normalized eigenvectors of A[][]
n	n can vary up to and including 10 but the matrices A and eigvec must have 10 columns.

Definition at line 244 of file matrix.c.

5.36.2.3 fEigenCompute4()

```
void fEigenCompute4 (
            float A[ ][4],
            float eigval[ ],
            float eigvec[ ][4],
            int8 n )
```

function computes all eigenvalues and eigenvectors of a real symmetric matrix A[0..n-1][0..n-1] stored in the top left of a 4x4 array A[4][4] A[][] is changed on output. The eigenvectors are not sorted by value. This function is identical to eigencompute10 except for the workaround for 4x4 matrices since C cannot handle functions accepting matrices with variable numbers of columns.

Parameters

eigval	eigval[0..n-1] returns the eigenvalues of A[][].
eigvec	eigvec[0..n-1][0..n-1] returns the normalized eigenvectors of A[][]
n	n can vary up to and including 4 but the matrices A and eigvec must have 4 columns.

Definition at line 417 of file matrix.c.

5.36.2.4 fmatrixAeqI()

```
void fmatrixAeqI (
            float * A[ ],
            int16 rc )
```

function sets the matrix A to the identity matrix

Parameters

A	pointer to the matrix
rc	dimension of the matrix

Definition at line 91 of file matrix.c.

5.36.2.5 fmatrixAeqInvA()

```
void fmatrixAeqInvA (
            float * A[],
            int8 iColInd[],
            int8 iRowInd[],
            int8 iPivot[],
            int8 isize,
            int8 * pierror )
```

function uses Gauss-Jordan elimination to compute the inverse of matrix A in situ on exit, A is replaced with its inverse

Definition at line 676 of file matrix.c.

5.36.2.6 fVeq3x3AxV()

```
void fVeq3x3AxV (
            float V[3],
            float A[][3] )
```

function multiplies the 3x1 vector V by a 3x3 matrix A

Parameters

V	used for both input and output

Definition at line 881 of file matrix.c.

5.36.2.7 fveqRu()

```
void fveqRu (
            float fv[],
            float fR[][3],
            float fu[],
            int8 itranspose )
```

function rotates 3x1 vector u onto 3x1 vector using 3x3 rotation matrix fR. the rotation is applied in the inverse direction if itranpose is true

Parameters

fv	3x1 output vector
fR	rotation matrix
fu	3x1 input vector
itranspose	true if inverse direction desired

Definition at line 830 of file matrix.c.

5.37 motionCheck.c File Reference

check to see if the board is moving.

```
#include "sensor_fusion.h"
```

Functions

- bool motionCheck (float sample[3], float baseline[3], float tolerance, uint32_t winLength, uint32_t *count)

5.37.1 Detailed Description

check to see if the board is moving.

This function would normally be called from your fusion_tasks in your main.c. See main_freertos_two_tasks_power_cycling.c for example usage.

5.37.2 Function Documentation

5.37.2.1 motionCheck()

```
bool motionCheck (
            float sample[3],
            float baseline[3],
            float tolerance,
            uint32_t winLength,
            uint32_t * count )
```

The motionCheck() function is not a sensor fusion function. It is a function that simply monitors an accelerometer or magnetometer tri-axial sensor output, returning Boolean true if the sensor appears to be stationary, and false otherwise. This function would normally be called from your fusion_tasks in your main().

Parameters

sample	processed triaxial sensor sample (accel or mag)
baseline	previous value to compare to
tolerance	how much tolerance you can stand
winLength	how many samples need to be stable to assert "noMotion"
count	how many samples so far we've been not moving

Definition at line 45 of file motionCheck.c.

5.38 orientation.c File Reference

Functions to convert between various orientation representations.

```
#include "stdio.h"
#include "math.h"
#include "stdlib.h"
#include "time.h"
#include "string.h"
#include "anomaly_detection.h"
#include "orientation.h"
#include "fusion.h"
#include "matrix.h"
#include "approximations.h"
```

Macros

- #define **SMALLQ0** 1E-4F
- #define **CORRUPTQUAT** 0.001F
- #define **SMALLMODULUS** 0.01F

Functions

- void fNEDAnglesDegFromRotationMatrix (float R[ ][3], float ∗pfPhiDeg, float ∗pfTheDeg, float ∗pfPsiDeg, float ∗pfRhoDeg, float ∗pfChiDeg)

    extract the NED angles in degrees from the NED rotation matrix
- void fAndroidAnglesDegFromRotationMatrix (float R[ ][3], float ∗pfPhiDeg, float ∗pfTheDeg, float ∗pfPsiDeg, float ∗pfRhoDeg, float ∗pfChiDeg)

    extract the Android angles in degrees from the Android rotation matrix
- void fWin8AnglesDegFromRotationMatrix (float R[ ][3], float ∗pfPhiDeg, float ∗pfTheDeg, float ∗pfPsiDeg, float ∗pfRhoDeg, float ∗pfChiDeg)

    extract the Windows 8 angles in degrees from the Windows 8 rotation matrix
- void fQuaternionFromRotationVectorDeg (Quaternion ∗pq, const float rvecdeg[ ], float fscaling)

    computes normalized rotation quaternion from a rotation vector (deg)
- void fQuaternionFromRotationMatrix (float R[ ][3], Quaternion ∗pq)

    compute the orientation quaternion from a 3x3 rotation matrix
- void fRotationMatrixFromQuaternion (float R[ ][3], const Quaternion ∗pq)

    compute the rotation matrix from an orientation quaternion
- void fRotationVectorDegFromQuaternion (Quaternion ∗pq, float rvecdeg[ ])

    computes rotation vector (deg) from rotation quaternion
- void fLPFOrientationQuaternion (Quaternion ∗pq, Quaternion ∗pLPq, float flpf, float fdeltat, float fOmega[ ])

    function low pass filters an orientation quaternion and computes virtual gyro rotation rate
- void qAeqBxC (Quaternion ∗pqA, const Quaternion ∗pqB, const Quaternion ∗pqC)

    function compute the quaternion product qB ∗ qC
- void qAeqAxB (Quaternion ∗pqA, const Quaternion ∗pqB)

    function compute the quaternion product qA = qA ∗ qB
- Quaternion qconjgAxB (const Quaternion ∗pqA, const Quaternion ∗pqB)

    function compute the quaternion product conjg(qA) ∗ qB
- void fqAeqNormqA (Quaternion ∗pqA)

    function normalizes a rotation quaternion and ensures q0 is non-negative
- void fqAeq1 (Quaternion ∗pqA)

    set a quaternion to the unit quaternion
- void fveqconjgquq (Quaternion ∗pfq, float fu[ ], float fv[ ])

5.38.1 Detailed Description

Functions to convert between various orientation representations.

Functions to convert between various orientation representations. Also includes functions for manipulating quaternions.

5.38.2 Function Documentation

5.38.2.1 fAndroidAnglesDegFromRotationMatrix()

```
void fAndroidAnglesDegFromRotationMatrix (
            float R[ ][3],
            float * pfPhiDeg,
            float * pfTheDeg,
            float * pfPsiDeg,
            float * pfRhoDeg,
            float * pfChiDeg )
```

extract the Android angles in degrees from the Android rotation matrix

Parameters

R	rotation matrix input
pfPhiDeg	the roll angle -90.0 $<=$ Phi $<=$ 90.0 deg
pfTheDeg	the pitch angle -180.0 $<=$ The $<$ 180.0 deg
pfPsiDeg	yaw angle Psi with range 0.0 $<=$ Psi $<$ 360.0 deg
pfRhoDeg	the compass heading angle Rho equals the yaw angle Psi
pfChiDeg	the tilt angle from vertical Chi (0 $<=$ Chi $<=$ 180 deg)

Definition at line 570 of file orientation.c.

5.38.2.2 fNEDAnglesDegFromRotationMatrix()

```
void fNEDAnglesDegFromRotationMatrix (
            float R[ ][3],
            float * pfPhiDeg,
            float * pfTheDeg,
            float * pfPsiDeg,
            float * pfRhoDeg,
            float * pfChiDeg )
```

extract the NED angles in degrees from the NED rotation matrix

Parameters

R	rotation matrix input
pfPhiDeg	output: the roll angle range -180.0 <= Phi < 180.0 deg
pfTheDeg	output: the pitch angle -90.0 <= Theta <= 90.0 deg
pfPsiDeg	output: the yaw (compass) angle 0.0 <= Psi < 360.0 deg
pfRhoDeg	output: For NED, the compass heading Rho equals the yaw angle Psi
pfChiDeg	output: the tilt angle from vertical Chi (0 <= Chi <= 180 deg)

Definition at line 514 of file orientation.c.

5.38.2.3 fQuaternionFromRotationMatrix()

```
void fQuaternionFromRotationMatrix (
            float R[ ][3],
            Quaternion * pq )
```

compute the orientation quaternion from a 3x3 rotation matrix

Parameters

R	Rotation matrix (input)
pq	Quaternion (output)

Definition at line 787 of file orientation.c.

5.38.2.4 fQuaternionFromRotationVectorDeg()

```
void fQuaternionFromRotationVectorDeg (
            Quaternion * pq,
            const float rvecdeg[ ],
            float fscaling )
```

computes normalized rotation quaternion from a rotation vector (deg)

Parameters

pq	quaternion (output)
rvecdeg	rotation vector in degrees
fscaling	delta Time

Definition at line 719 of file orientation.c.

5.38.2.5 fRotationMatrixFromQuaternion()

```
void fRotationMatrixFromQuaternion (
            float R[ ][3],
            const Quaternion * pq )
```

compute the rotation matrix from an orientation quaternion

Parameters

R	Rotation matrix (output)
pq	Quaternion (input)

Definition at line 828 of file orientation.c.

5.38.2.6 fRotationVectorDegFromQuaternion()

```
void fRotationVectorDegFromQuaternion (
            Quaternion * pq,
            float rvecdeg[ ] )
```

computes rotation vector (deg) from rotation quaternion

Parameters

pq	quaternion (input)
rvecdeg	rotation vector in degrees (output)

Definition at line 868 of file orientation.c.

5.38.2.7 fveqconjgquq()

```
void fveqconjgquq (
            Quaternion * pfq,
            float fu[ ],
            float fv[ ] )
```

function computes the rotation quaternion that rotates unit vector u onto unit vector v as v=q∗.u.q using q = 1/sqrt(2) ∗ {sqrt(1 + u.v) - u x v / sqrt(1 + u.v)}

Definition at line 1050 of file orientation.c.

5.38.2.8 fWin8AnglesDegFromRotationMatrix()

```
void fWin8AnglesDegFromRotationMatrix (
            float R[ ][3],
            float * pfPhiDeg,
            float * pfTheDeg,
            float * pfPsiDeg,
            float * pfRhoDeg,
            float * pfChiDeg )
```

extract the Windows 8 angles in degrees from the Windows 8 rotation matrix

Parameters

R	rotation matrix input
pfPhiDeg	the roll angle -90.0 <= Phi <= 90.0 deg
pfTheDeg	pitch angle Theta in the range -180.0 <= The < 180.0 deg
pfPsiDeg	yaw angle Psi in range 0.0 <= Psi < 360.0 deg
pfRhoDeg	the compass angle Rho = 360 - Psi
pfChiDeg	tilt angle from vertical Chi (0 <= Chi <= 180 deg)

Definition at line 627 of file orientation.c.

5.39 orientation.h File Reference

Functions to convert between various orientation representations.

Data Structures

- struct Quaternion

  quaternion structure definition

Typedefs

- typedef struct Quaternion Quaternion

  quaternion structure definition

Functions

- void f3DOFTiltNED (float fR[ ][3], float fGc[ ])

  Aerospace NED accelerometer 3DOF tilt function, computing rotation matrix fR.
- void f3DOFTiltAndroid (float fR[ ][3], float fGc[ ])

  Android accelerometer 3DOF tilt function computing, rotation matrix fR.
- void f3DOFTiltWin8 (float fR[ ][3], float fGc[ ])

  Windows 8 accelerometer 3DOF tilt function computing, rotation matrix fR.
- void f3DOFMagnetometerMatrixNED (float fR[ ][3], float fBc[ ])

  Aerospace NED magnetometer 3DOF flat eCompass function, computing rotation matrix fR.

- void f3DOFMagnetometerMatrixAndroid (float fR[ ][3], float fBc[ ])

  Android magnetometer 3DOF flat eCompass function, computing rotation matrix fR.
- void f3DOFMagnetometerMatrixWin8 (float fR[ ][3], float fBc[ ])

  Windows 8 magnetometer 3DOF flat eCompass function, computing rotation matrix fR.
- void feCompassNED (float fR[ ][3], float ∗pfDelta, float ∗pfsinDelta, float ∗pfcosDelta, float fBc[ ], float fGc[ ], float ∗pfmodBc, float ∗pfmodGc)

  NED: basic 6DOF e-Compass function, computing rotation matrix fR and magnetic inclination angle fDelta.
- void feCompassAndroid (float fR[ ][3], float ∗pfDelta, float ∗pfsinDelta, float ∗pfcosDelta, float fBc[ ], float f←Gc[ ], float ∗pfmodBc, float ∗pfmodGc)

  Android: basic 6DOF e-Compass function, computing rotation matrix fR and magnetic inclination angle fDelta.
- void feCompassWin8 (float fR[ ][3], float ∗pfDelta, float ∗pfsinDelta, float ∗pfcosDelta, float fBc[ ], float fGc[ ], float ∗pfmodBc, float ∗pfmodGc)

  Win8: basic 6DOF e-Compass function, computing rotation matrix fR and magnetic inclination angle fDelta.
- void fNEDAnglesDegFromRotationMatrix (float R[ ][3], float ∗pfPhiDeg, float ∗pfTheDeg, float ∗pfPsiDeg, float ∗pfRhoDeg, float ∗pfChiDeg)

  extract the NED angles in degrees from the NED rotation matrix
- void fAndroidAnglesDegFromRotationMatrix (float R[ ][3], float ∗pfPhiDeg, float ∗pfTheDeg, float ∗pfPsiDeg, float ∗pfRhoDeg, float ∗pfChiDeg)

  extract the Android angles in degrees from the Android rotation matrix
- void fWin8AnglesDegFromRotationMatrix (float R[ ][3], float ∗pfPhiDeg, float ∗pfTheDeg, float ∗pfPsiDeg, float ∗pfRhoDeg, float ∗pfChiDeg)

  extract the Windows 8 angles in degrees from the Windows 8 rotation matrix
- void fQuaternionFromRotationMatrix (float R[ ][3], Quaternion ∗pq)

  compute the orientation quaternion from a 3x3 rotation matrix
- void fRotationMatrixFromQuaternion (float R[ ][3], const Quaternion ∗pq)

  compute the rotation matrix from an orientation quaternion
- void qAeqBxC (Quaternion ∗pqA, const Quaternion ∗pqB, const Quaternion ∗pqC)

  function compute the quaternion product qB ∗ qC
- void qAeqAxB (Quaternion ∗pqA, const Quaternion ∗pqB)

  function compute the quaternion product qA = qA ∗ qB
- Quaternion qconjgAxB (const Quaternion ∗pqA, const Quaternion ∗pqB)

  function compute the quaternion product conjg(qA) ∗ qB
- void fqAeqNormqA (Quaternion ∗pqA)

  function normalizes a rotation quaternion and ensures q0 is non-negative
- void fqAeq1 (Quaternion ∗pqA)

  set a quaternion to the unit quaternion
- void fQuaternionFromRotationVectorDeg (Quaternion ∗pq, const float rvecdeg[ ], float fscaling)

  computes normalized rotation quaternion from a rotation vector (deg)
- void fRotationVectorDegFromQuaternion (Quaternion ∗pq, float rvecdeg[ ])

  computes rotation vector (deg) from rotation quaternion
- void fLPFOrientationQuaternion (Quaternion ∗pq, Quaternion ∗pLPq, float flpf, float fdeltat, float fOmega[ ])

  function low pass filters an orientation quaternion and computes virtual gyro rotation rate
- void fveqconjgquq (Quaternion ∗pfq, float fu[ ], float fv[ ])

5.39.1 Detailed Description

Functions to convert between various orientation representations.

Functions to convert between various orientation representations. Also includes functions for manipulating quaternions.

5.39.2 Function Documentation

5.39.2.1 f3DOFMagnetometerMatrixAndroid()

```
void f3DOFMagnetometerMatrixAndroid (
            float fR[ ][3],
            float fBc[ ] )
```

Android magnetometer 3DOF flat eCompass function, computing rotation matrix fR.

Parameters

fR	computed rotation matrix (output)
fBc	calibrated magnetometer reading (input)

5.39.2.2 f3DOFMagnetometerMatrixNED()

```
void f3DOFMagnetometerMatrixNED (
            float fR[ ][3],
            float fBc[ ] )
```

Aerospace NED magnetometer 3DOF flat eCompass function, computing rotation matrix fR.

Parameters

fR	computed rotation matrix (output)
fBc	calibrated magnetometer reading (input)

5.39.2.3 f3DOFMagnetometerMatrixWin8()

```
void f3DOFMagnetometerMatrixWin8 (
            float fR[ ][3],
            float fBc[ ] )
```

Windows 8 magnetometer 3DOF flat eCompass function, computing rotation matrix fR.

Parameters

fR	computed rotation matrix (output)
fBc	calibrated magnetometer reading (input)

5.39.2.4 f3DOFTiltAndroid()

```
void f3DOFTiltAndroid (
            float fR[ ][3],
            float fGc[ ] )
```

Android accelerometer 3DOF tilt function computing, rotation matrix fR.

Parameters

fR	computed rotation matrix (output)
fGc	calibrated accelerometer input vector

5.39.2.5 f3DOFTiltNED()

```
void f3DOFTiltNED (
            float fR[ ][3],
            float fGc[ ] )
```

Aerospace NED accelerometer 3DOF tilt function, computing rotation matrix fR.

Parameters

fR	computed rotation matrix (output)
fGc	calibrated accelerometer input vector

5.39.2.6 f3DOFTiltWin8()

```
void f3DOFTiltWin8 (
            float fR[ ][3],
            float fGc[ ] )
```

Windows 8 accelerometer 3DOF tilt function computing, rotation matrix fR.

Parameters

fR	computed rotation matrix (output)
fGc	calibrated accelerometer input vector

5.39.2.7 fAndroidAnglesDegFromRotationMatrix()

```
void fAndroidAnglesDegFromRotationMatrix (
            float R[ ][3],
            float * pfPhiDeg,
            float * pfTheDeg,
            float * pfPsiDeg,
            float * pfRhoDeg,
            float * pfChiDeg )
```

extract the Android angles in degrees from the Android rotation matrix

Parameters

R	rotation matrix input
pfPhiDeg	the roll angle -90.0 $<=$ Phi $<=$ 90.0 deg
pfTheDeg	the pitch angle -180.0 $<=$ The $<$ 180.0 deg
pfPsiDeg	yaw angle Psi with range 0.0 $<=$ Psi $<$ 360.0 deg
pfRhoDeg	the compass heading angle Rho equals the yaw angle Psi
pfChiDeg	the tilt angle from vertical Chi (0 $<=$ Chi $<=$ 180 deg)

Definition at line 570 of file orientation.c.

5.39.2.8 feCompassAndroid()

```
void feCompassAndroid (
            float fR[ ][3],
            float * pfDelta,
            float * pfsinDelta,
            float * pfcosDelta,
            float fBc[ ],
            float fGc[ ],
            float * pfmodBc,
            float * pfmodGc )
```

Android: basic 6DOF e-Compass function, computing rotation matrix fR and magnetic inclination angle fDelta.

Parameters

fR	computed rotation matrix (output)
pfDelta	magnetic inclination angle (output)
pfsinDelta	sin of the inclination angle
pfcosDelta	cos of the inclination angle
fBc	calibrated magnetometer reading (input)
fGc	calibrated accelerometer input vector (input)
pfmodBc	modulus of the calibrated magnetic vector
pfmodGc	modulus of the calibrated accelerometer vector

5.39.2.9 feCompassNED()

```
void feCompassNED (
            float fR[][3],
            float * pfDelta,
            float * pfsinDelta,
            float * pfcosDelta,
            float fBc[],
            float fGc[],
            float * pfmodBc,
            float * pfmodGc )
```

NED: basic 6DOF e-Compass function, computing rotation matrix fR and magnetic inclination angle fDelta.

Parameters

fR	computed rotation matrix (output)
pfDelta	magnetic inclination angle (output)
pfsinDelta	sin of the inclination angle
pfcosDelta	cos of the inclination angle
fBc	calibrated magnetometer vector (input)
fGc	calibrated accelerometer input vector (input)
pfmodBc	modulus of the calibrated magnetic vector
pfmodGc	modulus of the calibrated accelerometer vector

5.39.2.10 feCompassWin8()

```
void feCompassWin8 (
            float fR[][3],
            float * pfDelta,
            float * pfsinDelta,
            float * pfcosDelta,
            float fBc[],
            float fGc[],
            float * pfmodBc,
            float * pfmodGc )
```

Win8: basic 6DOF e-Compass function, computing rotation matrix fR and magnetic inclination angle fDelta.

Parameters

fR	computed rotation matrix (output)
pfDelta	magnetic inclination angle (output)
pfsinDelta	sin of the inclination angle
pfcosDelta	cos of the inclination angle
fBc	calibrated magnetometer reading (input)
fGc	calibrated accelerometer input vector (input)
pfmodBc	modulus of the calibrated magnetic vector
pfmodGc	modulus of the calibrated accelerometer vector

5.39.2.11 fNEDAnglesDegFromRotationMatrix()

```
void fNEDAnglesDegFromRotationMatrix (
            float R[ ][3],
            float * pfPhiDeg,
            float * pfTheDeg,
            float * pfPsiDeg,
            float * pfRhoDeg,
            float * pfChiDeg )
```

extract the NED angles in degrees from the NED rotation matrix

Parameters

R	rotation matrix input
pfPhiDeg	output: the roll angle range -180.0 <= Phi < 180.0 deg
pfTheDeg	output: the pitch angle -90.0 <= Theta <= 90.0 deg
pfPsiDeg	output: the yaw (compass) angle 0.0 <= Psi < 360.0 deg
pfRhoDeg	output: For NED, the compass heading Rho equals the yaw angle Psi
pfChiDeg	output: the tilt angle from vertical Chi (0 <= Chi <= 180 deg)

Definition at line 514 of file orientation.c.

5.39.2.12 fQuaternionFromRotationMatrix()

```
void fQuaternionFromRotationMatrix (
            float R[ ][3],
            Quaternion * pq )
```

compute the orientation quaternion from a 3x3 rotation matrix

Parameters

R	Rotation matrix (input)
pq	Quaternion (output)

Definition at line 787 of file orientation.c.

5.39.2.13 fQuaternionFromRotationVectorDeg()

```
void fQuaternionFromRotationVectorDeg (
            Quaternion * pq,
```

```
const float rvecdeg[],
float fscaling )
```

computes normalized rotation quaternion from a rotation vector (deg)

Parameters

pq	quaternion (output)
rvecdeg	rotation vector in degrees
fscaling	delta Time

Definition at line 719 of file orientation.c.

5.39.2.14 fRotationMatrixFromQuaternion()

```
void fRotationMatrixFromQuaternion (
            float R[ ][3],
            const Quaternion * pq )
```

compute the rotation matrix from an orientation quaternion

Parameters

R	Rotation matrix (output)
pq	Quaternion (input)

Definition at line 828 of file orientation.c.

5.39.2.15 fRotationVectorDegFromQuaternion()

```
void fRotationVectorDegFromQuaternion (
            Quaternion * pq,
            float rvecdeg[ ] )
```

computes rotation vector (deg) from rotation quaternion

Parameters

pq	quaternion (input)
rvecdeg	rotation vector in degrees (output)

Definition at line 868 of file orientation.c.

5.39.2.16 fveqconjgquq()

```
void fveqconjgquq (
            Quaternion * pfq,
```

```
              float fu[],
              float fv[] )
```

function computes the rotation quaternion that rotates unit vector u onto unit vector v as v=q∗.u.q using q = 1/sqrt(2) ∗ {sqrt(1 + u.v) - u x v / sqrt(1 + u.v)}

Definition at line 1050 of file orientation.c.

5.39.2.17 fWin8AnglesDegFromRotationMatrix()

```
void fWin8AnglesDegFromRotationMatrix (
              float R[][3],
              float * pfPhiDeg,
              float * pfTheDeg,
              float * pfPsiDeg,
              float * pfRhoDeg,
              float * pfChiDeg )
```

extract the Windows 8 angles in degrees from the Windows 8 rotation matrix

Parameters

R	rotation matrix input
pfPhiDeg	the roll angle -90.0 <= Phi <= 90.0 deg
pfTheDeg	pitch angle Theta in the range -180.0 <= The < 180.0 deg
pfPsiDeg	yaw angle Psi in range 0.0 <= Psi < 360.0 deg
pfRhoDeg	the compass angle Rho = 360 - Psi
pfChiDeg	tilt angle from vertical Chi (0 <= Chi <= 180 deg)

Definition at line 627 of file orientation.c.

5.40 output_stream.c File Reference

The output_stream.c file implements the streaming packets functionalities for ADT. Real-time features, anomaly detection results, trained models such GMM and OC-SVM information are transtmitted.

```
#include "stdlib.h"
#include "FreeRTOS.h"
#include "task.h"
#include "queue.h"
#include "timers.h"
#include "event_groups.h"
#include "fsl_debug_console.h"
#include "board.h"
#include "pin_mux.h"
#include "clock_config.h"
#include "fsl_port.h"
#include "fsl_i2c.h"
#include "Driver_USART.h"
```

```
#include "fsl_i2c_cmsis.h"
#include "fsl_uart_cmsis.h"
#include "issdk_hal.h"
#include "fxas21002.h"
#include "fxos8700.h"
#include "register_io_i2c.h"
#include "host_io_uart.h"
#include "Driver_GPIO.h"
#include "gpio_driver.h"
#include "anomaly_detection.h"
#include "control.h"
#include "status.h"
#include "drivers.h"
#include "output_stream.h"
```

Macros

- #define **PASSFAIL** 0x80
- #define **TRAINRUN** 0x40

Functions

- void send_model_result (bool passFail, bool trainRun, uint8_t modelNumber, uint16_t featureInterval, float feature1, float feature2)

    This function is called when stream packets are transmitted to GUI with the current feature samples and detection restuls.
- void send_computed_gmm_model (ModelInstance ∗modelInstance, uint16_t featureInterval)

    This function is called when stream packets are transmitted to GUI with the current GMM information.
- void send_computed_svm_model (ModelInstance ∗modelInstance, uint16_t featureInterval)

    This function is called when stream packets are transmitted to GUI with the current OC-SVM information.
- void streamFeatures ()

    This function is called when stream packets are transmitted to GUI with features and anomaly detection results.
- bool streamModels (struct ModelCollection ∗modelCollection, uint8_t modelID)

    This function is called when stream packets are transmitted to GUI with model information.

Variables

- Globals gbls

    This is the primary sensor fusion data structure.

5.40.1 Detailed Description

The output_stream.c file implements the streaming packets functionalities for ADT. Real-time features, anomaly detection results, trained models such GMM and OC-SVM information are transtmitted.

5.40.2 Function Documentation

5.40.2.1 send_computed_gmm_model()

```
void send_computed_gmm_model (
            ModelInstance * modelInstance,
            uint16_t featureInterval )
```

This function is called when stream packets are transmitted to GUI with the current GMM information.

Details about the packet structure is available in ADT manual.

Definition at line 106 of file output_stream.c.

5.40.2.2 send_computed_svm_model()

```
void send_computed_svm_model (
            ModelInstance * modelInstance,
            uint16_t featureInterval )
```

This function is called when stream packets are transmitted to GUI with the current OC-SVM information.

Details about the packet structure is available in ADT manual.

Definition at line 152 of file output_stream.c.

5.40.2.3 send_model_result()

```
void send_model_result (
            bool passFail,
            bool trainRun,
            uint8_t modelNumber,
            uint16_t featureInterval,
            float feature1,
            float feature2 )
```

This function is called when stream packets are transmitted to GUI with the current feature samples and detection restuls.

Details about the packet structure is available in ADT manual.

Definition at line 82 of file output_stream.c.

5.40.2.4 streamFeatures()

```
void streamFeatures ( )
```

This function is called when stream packets are transmitted to GUI with features and anomaly detection results.

Details about the packet structure is available in ADT manual.

Definition at line 209 of file output_stream.c.

5.40.2.5 streamModels()

```
bool streamModels (
            struct ModelCollection * modelCollection,
            uint8_t modelID )
```

This function is called when stream packets are transmitted to GUI with model information.

Details about the packet structure is available in ADT manual.

Definition at line 256 of file output_stream.c.

5.41 precisionAccelerometer.c File Reference

Implements accelerometer calibration routines.

```
#include <stdio.h>
#include "anomaly_detection.h"
#include "fusion.h"
```

Functions

- void fInitializeAccelCalibration (struct AccelCalibration *pthisAccelCal, struct AccelBuffer *pthisAccelBuffer, volatile int8 *AccelCalPacketOn)

  Initialize the accelerometer calibration functions.
- void fUpdateAccelBuffer (struct AccelCalibration *pthisAccelCal, struct AccelBuffer *pthisAccelBuffer, struct AccelSensor *pthisAccel, volatile int8 *AccelCalPacketOn)

  Update the buffer used to store samples used for accelerometer calibration.
- void fInvertAccelCal (struct AccelSensor *pthisAccel, struct AccelCalibration *pthisAccelCal)

  function maps the accelerometer data fSum (g) onto precision calibrated and de-rotated data fGc (g), iGc (counts)
- void fRunAccelCalibration (struct AccelCalibration *pthisAccelCal, struct AccelBuffer *pthisAccelBuffer, struct AccelSensor *pthisAccel)

  function runs the precision accelerometer calibration
- void fComputeAccelCalibration4 (struct AccelBuffer *pthisAccelBuffer, struct AccelCalibration *pthisAccel←Cal, struct AccelSensor *pthisAccel)

  calculate the 4 element calibration from the available measurements
- void fComputeAccelCalibration7 (struct AccelBuffer *pthisAccelBuffer, struct AccelCalibration *pthisAccel←Cal, struct AccelSensor *pthisAccel)

  calculate the 7 element calibration from the available measurements
- void fComputeAccelCalibration10 (struct AccelBuffer *pthisAccelBuffer, struct AccelCalibration *pthisAccel←Cal, struct AccelSensor *pthisAccel)

  calculate the 10 element calibration from the available measurements

5.41.1 Detailed Description

Implements accelerometer calibration routines.

5.41.2 Function Documentation

5.41.2.1 fComputeAccelCalibration10()

```
void fComputeAccelCalibration10 (
            struct AccelBuffer * pthisAccelBuffer,
            struct AccelCalibration * pthisAccelCal,
            struct AccelSensor * pthisAccel )
```

calculate the 10 element calibration from the available measurements

Parameters

pthisAccelBuffer	Buffer of measurements used as input to the accel calibration functions
pthisAccelCal	Accelerometer calibration parameter structure
pthisAccel	Pointer to the accelerometer input/state structure

Definition at line 80 of file precisionAccelerometer.c.

5.41.2.2 fComputeAccelCalibration4()

```
void fComputeAccelCalibration4 (
            struct AccelBuffer * pthisAccelBuffer,
            struct AccelCalibration * pthisAccelCal,
            struct AccelSensor * pthisAccel )
```

calculate the 4 element calibration from the available measurements

Parameters

pthisAccelBuffer	Buffer of measurements used as input to the accel calibration functions
pthisAccelCal	Accelerometer calibration parameter structure
pthisAccel	Pointer to the accelerometer input/state structure

Definition at line 66 of file precisionAccelerometer.c.

5.41.2.3 fComputeAccelCalibration7()

```
void fComputeAccelCalibration7 (
            struct AccelBuffer * pthisAccelBuffer,
            struct AccelCalibration * pthisAccelCal,
            struct AccelSensor * pthisAccel )
```

calculate the 7 element calibration from the available measurements

Parameters

pthisAccelBuffer	Buffer of measurements used as input to the accel calibration functions
pthisAccelCal	Accelerometer calibration parameter structure
pthisAccel	Pointer to the accelerometer input/state structure

Definition at line 73 of file precisionAccelerometer.c.

5.41.2.4 fInitializeAccelCalibration()

```
void fInitializeAccelCalibration (
            struct AccelCalibration * pthisAccelCal,
            struct AccelBuffer * pthisAccelBuffer,
            volatile int8 * AccelCalPacketOn )
```

Initialize the accelerometer calibration functions.

Parameters

pthisAccelCal	Accelerometer calibration parameter structure
pthisAccelBuffer	Buffer of measurements used as input to the accel calibration functions
AccelCalPacketOn	Used to coordinate calibration sample storage and communications

Definition at line 40 of file precisionAccelerometer.c.

5.41.2.5 fInvertAccelCal()

```
void fInvertAccelCal (
            struct AccelSensor * pthisAccel,
            struct AccelCalibration * pthisAccelCal )
```

function maps the accelerometer data fSum (g) onto precision calibrated and de-rotated data fGc (g), iGc (counts)

Parameters

pthisAccel	Pointer to the accelerometer input/state structure
pthisAccelCal	Accelerometer calibration parameter structure

Definition at line 53 of file precisionAccelerometer.c.

5.41.2.6 fRunAccelCalibration()

```
void fRunAccelCalibration (
            struct AccelCalibration * pthisAccelCal,
```

```
            struct AccelBuffer * pthisAccelBuffer,
            struct AccelSensor * pthisAccel )
```

function runs the precision accelerometer calibration

Parameters

pthisAccelCal	Accelerometer calibration parameter structure
pthisAccelBuffer	Buffer of measurements used as input to the accel calibration functions
pthisAccel	Pointer to the accelerometer input/state structure

Definition at line 59 of file precisionAccelerometer.c.

5.41.2.7 fUpdateAccelBuffer()

```
void fUpdateAccelBuffer (
            struct AccelCalibration * pthisAccelCal,
            struct AccelBuffer * pthisAccelBuffer,
            struct AccelSensor * pthisAccel,
            volatile int8 * AccelCalPacketOn )
```

Update the buffer used to store samples used for accelerometer calibration.

Parameters

pthisAccelCal	Accelerometer calibration parameter structure
pthisAccelBuffer	Buffer of measurements used as input to the accel calibration functions
pthisAccel	Pointer to the accelerometer input/state structure
AccelCalPacketOn	Used to coordinate calibration sample storage and communications

Definition at line 46 of file precisionAccelerometer.c.

5.42 precisionAccelerometer.h File Reference

Implements accelerometer calibration routines.

Data Structures

- struct AccelBuffer

  accelerometer measurement buffer
- struct AccelCalibration

  precision accelerometer calibration structure

Macros

- #define ACCEL_CAL_AVERAGING_SECS 2

    calibration constants
- #define MAX_ACCEL_CAL_ORIENTATIONS 12

    number of stored precision accelerometer measurements

Typedefs

- typedef struct AccelBuffer AccelBuffer

    accelerometer measurement buffer
- typedef struct AccelCalibration AccelCalibration

    precision accelerometer calibration structure

Functions

- void fInitializeAccelCalibration (struct AccelCalibration ∗pthisAccelCal, struct AccelBuffer ∗pthisAccelBuffer, volatile int8_t ∗AccelCalPacketOn)

    Initialize the accelerometer calibration functions.
- void fUpdateAccelBuffer (struct AccelCalibration ∗pthisAccelCal, struct AccelBuffer ∗pthisAccelBuffer, struct AccelSensor ∗pthisAccel, volatile int8_t ∗AccelCalPacketOn)

    Update the buffer used to store samples used for accelerometer calibration.
- void fInvertAccelCal (struct AccelSensor ∗pthisAccel, struct AccelCalibration ∗pthisAccelCal)

    function maps the accelerometer data fSum (g) onto precision calibrated and de-rotated data fGc (g), iGc (counts)
- void fRunAccelCalibration (struct AccelCalibration ∗pthisAccelCal, struct AccelBuffer ∗pthisAccelBuffer, struct AccelSensor ∗pthisAccel)

    function runs the precision accelerometer calibration
- void fComputeAccelCalibration4 (struct AccelBuffer ∗pthisAccelBuffer, struct AccelCalibration ∗pthisAccel↔Cal, struct AccelSensor ∗pthisAccel)

    calculate the 4 element calibration from the available measurements
- void fComputeAccelCalibration7 (struct AccelBuffer ∗pthisAccelBuffer, struct AccelCalibration ∗pthisAccel↔Cal, struct AccelSensor ∗pthisAccel)

    calculate the 7 element calibration from the available measurements
- void fComputeAccelCalibration10 (struct AccelBuffer ∗pthisAccelBuffer, struct AccelCalibration ∗pthisAccel↔Cal, struct AccelSensor ∗pthisAccel)

    calculate the 10 element calibration from the available measurements

5.42.1 Detailed Description

Implements accelerometer calibration routines.

5.42.2 Macro Definition Documentation

5.42.2.1 ACCEL_CAL_AVERAGING_SECS

```
#define ACCEL_CAL_AVERAGING_SECS 2
```

calibration constants

calibration measurement averaging period (s)

Definition at line 39 of file precisionAccelerometer.h.

5.42.3 Function Documentation

5.42.3.1 fComputeAccelCalibration10()

```
void fComputeAccelCalibration10 (
            struct AccelBuffer * pthisAccelBuffer,
            struct AccelCalibration * pthisAccelCal,
            struct AccelSensor * pthisAccel )
```

calculate the 10 element calibration from the available measurements

Parameters

pthisAccelBuffer	Buffer of measurements used as input to the accel calibration functions
pthisAccelCal	Accelerometer calibration parameter structure
pthisAccel	Pointer to the accelerometer input/state structure

Definition at line 80 of file precisionAccelerometer.c.

5.42.3.2 fComputeAccelCalibration4()

```
void fComputeAccelCalibration4 (
            struct AccelBuffer * pthisAccelBuffer,
            struct AccelCalibration * pthisAccelCal,
            struct AccelSensor * pthisAccel )
```

calculate the 4 element calibration from the available measurements

Parameters

pthisAccelBuffer	Buffer of measurements used as input to the accel calibration functions
pthisAccelCal	Accelerometer calibration parameter structure
pthisAccel	Pointer to the accelerometer input/state structure

Definition at line 66 of file precisionAccelerometer.c.

5.42.3.3 fComputeAccelCalibration7()

```
void fComputeAccelCalibration7 (
            struct AccelBuffer * pthisAccelBuffer,
            struct AccelCalibration * pthisAccelCal,
            struct AccelSensor * pthisAccel )
```

calculate the 7 element calibration from the available measurements

Parameters

pthisAccelBuffer	Buffer of measurements used as input to the accel calibration functions
pthisAccelCal	Accelerometer calibration parameter structure
pthisAccel	Pointer to the accelerometer input/state structure

Definition at line 73 of file precisionAccelerometer.c.

5.42.3.4 fInitializeAccelCalibration()

```
void fInitializeAccelCalibration (
            struct AccelCalibration * pthisAccelCal,
            struct AccelBuffer * pthisAccelBuffer,
            volatile int8_t * AccelCalPacketOn )
```

Initialize the accelerometer calibration functions.

Parameters

pthisAccelCal	Accelerometer calibration parameter structure
pthisAccelBuffer	Buffer of measurements used as input to the accel calibration functions
AccelCalPacketOn	Used to coordinate calibration sample storage and communications

Definition at line 40 of file precisionAccelerometer.c.

5.42.3.5 fInvertAccelCal()

```
void fInvertAccelCal (
            struct AccelSensor * pthisAccel,
            struct AccelCalibration * pthisAccelCal )
```

function maps the accelerometer data fSum (g) onto precision calibrated and de-rotated data fGc (g), iGc (counts)

Parameters

pthisAccel	Pointer to the accelerometer input/state structure
pthisAccelCal	Accelerometer calibration parameter structure

Definition at line 53 of file precisionAccelerometer.c.

5.42.3.6 fRunAccelCalibration()

```
void fRunAccelCalibration (
            struct AccelCalibration * pthisAccelCal,
            struct AccelBuffer * pthisAccelBuffer,
            struct AccelSensor * pthisAccel )
```

function runs the precision accelerometer calibration

Parameters

pthisAccelCal	Accelerometer calibration parameter structure
pthisAccelBuffer	Buffer of measurements used as input to the accel calibration functions
pthisAccel	Pointer to the accelerometer input/state structure

Definition at line 59 of file precisionAccelerometer.c.

5.42.3.7 fUpdateAccelBuffer()

```
void fUpdateAccelBuffer (
            struct AccelCalibration * pthisAccelCal,
            struct AccelBuffer * pthisAccelBuffer,
            struct AccelSensor * pthisAccel,
            volatile int8_t * AccelCalPacketOn )
```

Update the buffer used to store samples used for accelerometer calibration.

Parameters

pthisAccelCal	Accelerometer calibration parameter structure
pthisAccelBuffer	Buffer of measurements used as input to the accel calibration functions
pthisAccel	Pointer to the accelerometer input/state structure
AccelCalPacketOn	Used to coordinate calibration sample storage and communications

Definition at line 46 of file precisionAccelerometer.c.

5.43 process_host_command.c File Reference

The process_host_command.c file implements the embedded functional interfaces and host i/o interface.

```
#include "FreeRTOS.h"
#include "task.h"
#include "queue.h"
#include "timers.h"
#include "event_groups.h"
#include "fsl_debug_console.h"
#include "board.h"
#include "pin_mux.h"
#include "clock_config.h"
#include "fsl_port.h"
#include "fsl_i2c.h"
#include "Driver_USART.h"
#include "fsl_i2c_cmsis.h"
#include "fsl_uart_cmsis.h"
#include "issdk_hal.h"
#include "fxas21002.h"
#include "fxos8700.h"
#include "register_io_i2c.h"
#include "host_io_uart.h"
#include "Driver_GPIO.h"
#include "gpio_driver.h"
#include "anomaly_detection.h"
#include "control.h"
#include "status.h"
#include "drivers.h"
```

Macros

- #define **APPLICATION_NAME** "No Shield:Anomaly Detection:0.1" /∗ Orientation Application Name ∗/
- #define **DECODE_DIM**(X) ((X&0X10)>>4)
- #define **DECODE_DIM_SVM**(X) ((X&0x80)>>7)
- #define **DECODE_CLR**(X) (X&0x01)
- #define **DECODE_STOP**(X) ((X&0x02)>>1)
- #define **DECODE_RUN**(X) ((X&0x04)>>2)
- #define **DECODE_TRAIN**(X) ((X&0x08)>>3)
- #define **DECODE_DELETE**(X) ((X&0x10)>>4)
- #define **DECODE_DELETE_ALL**(X) ((X&0x20)>>5)
- #define **DECODE_TED**(X) ((X&0x40)>>6)
- #define **DECODE_STREAM_ENABLE**(X) ((X&0x80)>>7)
- #define **UPPER_NIBBLE**(X) (X>>4)
- #define **LOWER_NIBBLE**(X) (X&0x0F)
- #define **SHORT**(X, Y) ((X<<8) + Y)

Functions

- void **getEncodedSampleRates** (uint32_t ∗sampleRates)
- bool **setclr_feature** (sensor_t sensor, feature_t feature, axis_t axis, bool sts)
- bool **setFeatureInterval** (int32_t featureInterval)

- bool processSetModelModeCommand (uint8_t ModelType, uint8_t ModelNumber, bool StreamEnable, uint8_t command)

    This function is used to process model mode commands.
- bool processGmmConfigurationCommand (uint8_t ModelType, uint8_t ModelNumber, bool DIM, uint8_t MaxGaussians, uint16_t BufSize, uint16_t ComputeInterval, uint8_t ThresholdX100, feature_t FeatureCode1, feature_t FeatureCode2, sensor_t SensorCode1, sensor_t SensorCode2, axis_t AxisCode1, axis_t AxisCode2)

    This function is called when GMM configureation is processed.
- bool processOcSvmConfigurationCommand (uint8_t ModelType, uint8_t ModelNumber, uint8_t KernelType, bool DIM, uint16_t BufSize, uint16_t ComputeInterval, uint8_t nuX100, uint8_t gammaX100, feature_t FeatureCode1, feature_t FeatureCode2, sensor_t SensorCode1, sensor_t SensorCode2, axis_t AxisCode1, axis_t AxisCode2)

    This function is called when SVM configureation is processed.
- bool **processEnsembleConfigurationCommand** (uint8_t ModelType, uint8_t ModelNumber, uint8_t VotesRequired, uint8_t NumSubModels, uint8_t ∗SubModelSpecifier)
- bool **setclr_features** (sensor_t sensor, axis_t axis, byte ∗hostCommand, int idx)
- feature_t **featureEnumFromBitfield** (uint16_t bitfield)
- bool **process_host_command** (uint8_t tag, uint8_t ∗hostCommand, uint8_t ∗hostResponse, size_t ∗hostMsgSize, size_t respBufferSize)

Variables

- Globals gbls

    This is the primary sensor fusion data structure.
- int8_t **current_model_id** =0x00

5.43.1 Detailed Description

The process_host_command.c file implements the embedded functional interfaces and host i/o interface.

5.43.2 Function Documentation

5.43.2.1 processGmmConfigurationCommand()

```
bool processGmmConfigurationCommand (
            uint8_t ModelType,
            uint8_t ModelNumber,
            bool DIM,
            uint8_t MaxGaussians,
            uint16_t BufSize,
            uint16_t ComputeInterval,
            uint8_t ThresholdX100,
            feature_t FeatureCode1,
            feature_t FeatureCode2,
            sensor_t SensorCode1,
            sensor_t SensorCode2,
            axis_t AxisCode1,
            axis_t AxisCode2 )
```

This function is called when GMM configureation is processed.

The configuration is recieved from GUI and set to each model instance.

Parameters

ModelType	Model type
ModelNumber	Model ID.
DIM	1 or 2 dimension of feature samples.
MaxGaussians	The maximum number of Gaussian components in GMM usage.
BufSize	The size of moving window in feature buffer.
ComputeInterval	The rate of training. At every ComputeInterval, training is executed.
ThresholdX100	The treshold value assigned by a user. 100 times larger values is transmitted to use integer type rather than float and save the packet size.
FeatureCode1	The first dimension of feature type.
FeatureCode2	The second dimension of feature type.
SensorCode1	The first dimension of sensor type.
SensorCode2	The second dimension of sensor type.
AxisCode1	The axis of SensorCode1.
AxisCode2	The axis of SensorCode2.

Definition at line 175 of file process_host_command.c.

5.43.2.2 processOcSvmConfigurationCommand()

```
bool processOcSvmConfigurationCommand (
            uint8_t ModelType,
            uint8_t ModelNumber,
            uint8_t KernelType,
            bool DIM,
            uint16_t BufSize,
            uint16_t ComputeInterval,
            uint8_t nuX100,
            uint8_t gammaX100,
            feature_t FeatureCode1,
            feature_t FeatureCode2,
            sensor_t SensorCode1,
            sensor_t SensorCode2,
            axis_t AxisCode1,
            axis_t AxisCode2 )
```

This function is called when SVM configureation is processed.

The configuration is recieved from GUI and set to each model instance.

Parameters

ModelType	Model type
ModelNumber	Model ID.
DIM	1 or 2 dimension of feature samples.
MaxGaussians	The maximum number of Gaussian components in GMM usage.
BufSize	The size of moving window in feature buffer.
ComputeInterval	The rate of training. At every ComputeInterval, training is executed.

Parameters

nuX100	The nu parameter of one class SVM. It controls the bounds of how many SVs and threshold level. 100 times larger values is transmitted to use integer type rather than float and save the packet size.
gammaX100	The kernel size parameter assigned by a user. 100 times larger values is transmitted to use integer type rather than float and save the packet size.
FeatureCode1	The first dimension of feature type.
FeatureCode2	The second dimension of feature type.
SensorCode1	The first dimension of sensor type.
SensorCode2	The second dimension of sensor type.
AxisCode1	The axis of SensorCode1.
AxisCode2	The axis of SensorCode2.

Definition at line 239 of file process_host_command.c.

5.43.2.3 processSetModelModeCommand()

```
bool processSetModelModeCommand (
            uint8_t ModelType,
            uint8_t ModelNumber,
            bool StreamEnable,
            uint8_t command )
```

This function is used to process model mode commands.

With the received packet, model type, model ID, and appropriate command are set to a model instance.

Parameters

ModelType	Model types.
ModelNumber	Model ID.
StreamEnable	An indicator for enabling streaming.
command	The commands such as training and run.

Returns

true / false True if success.

Definition at line 123 of file process_host_command.c.

5.44 standard_build.h File Reference

A "standard" build configuration file.

Macros

- #define THISBUILD 700

    define build number sent in debug packet for display purposes only

CoordinateSystemBitFields

These defines determine the frame of reference (x, y, z axes and Euler angles) standard to be used for a particular build. Change THISCOORDSYSTEM to whichever of NED, ANDROID or WIN8 you prefer.

- #define NED 0

    identifier for NED (Aerospace) axes and angles
- #define ANDROID 1

    identifier for Android axes and angles
- #define WIN8 2

    identifier for Windows 8 axes and angles
- #define THISCOORDSYSTEM NED

    the coordinate system to be used

SensorBitFields

These bit-field values are used to declare which sensor types are used in the application. Change bit-field values to 0x0000 for any features NOT USED

- #define F_USING_ACCEL 0x0001

    nominally 0x0001 if an accelerometer is to be used, 0x0000 otherwise
- #define F_USING_MAG 0x0002

    nominally 0x0002 if an magnetometer is to be used, 0x0000 otherwise
- #define F_USING_GYRO 0x0004

    nominally 0x0004 if a gyro is to be used, 0x0000 otherwise
- #define F_USING_PRESSURE 0x0008

    nominally 0x0008 if altimeter is to be used, 0x0000 otherwise
- #define F_USING_TEMPERATURE 0x0010

    nominally 0x0010 if temp sensor is to be used, 0x0000 otherwise
- #define F_ALL_SENSORS 0x001F

    refers to all applicable sensor types for the given physical unit

FusionSelectionBitFields

These bit-field values are used to declare which sensor fusion algorithms are used in the application. You can use more than one, altough they all run from the same data. Change individual bit-field values to 0x0000 for any features NOT USED.

- #define F_1DOF_P_BASIC 0x0100

    1DOF pressure (altitude) and temperature algorithm selector - 0x0100 to include, 0x0000 otherwise
- #define F_3DOF_G_BASIC 0x0200

    3DOF accel tilt (accel) algorithm selector - 0x0200 to include, 0x0000 otherwise
- #define F_3DOF_B_BASIC 0x0400

    3DOF mag eCompass (vehicle/mag) algorithm selector - 0x0400 to include, 0x0000 otherwise
- #define F_3DOF_Y_BASIC 0x0800

    3DOF gyro integration algorithm selector - 0x0800 to include, 0x0000 otherwise
- #define F_6DOF_GB_BASIC 0x1000

    6DOF accel and mag eCompass algorithm selector - 0x1000 to include, 0x0000 otherwise
- #define F_6DOF_GY_KALMAN 0x2000

    6DOF accel and gyro (Kalman) algorithm selector - 0x2000 to include, 0x0000 otherwise
- #define F_9DOF_GBY_KALMAN 0x4000

    9DOF accel, mag and gyro algorithm selector - 0x4000 to include, 0x0000 otherwise

SensorParameters

FIFO sizes effect the size of the sensor data structures. ODR refers to "Output Data Rate"

- #define ACCEL_FIFO_SIZE 32
  FXOS8700 (accel), MMA8652, FXLS8952 all have 32 element FIFO.
- #define MAG_FIFO_SIZE 16
  FXOS8700 (mag), MAG3110 have no FIFO so equivalent to 1 element FIFO.
- #define GYRO_FIFO_SIZE 32
  FXAX21000, FXAS21002 have 32 element FIFO.
- #define ACCEL_ODR_HZ 200
  (int) requested accelerometer ODR Hz (over-rides MAG_ODR_HZ for FXOS8700)
- #define MAG_ODR_HZ 200
  (int) requested magnetometer ODR Hz (over-ridden by ACCEL_ODR_HZ for FXOS8700)
- #define GYRO_ODR_HZ 400
  (int) requested gyroscope ODR Hz
- #define FUSION_HZ 40
  (int) actual rate of fusion algorithm execution and sensor FIFO reads
- #define FAST_LOOP_HZ 80
  Over Sample Ratio ∗ FUSION_HZ when using no FIFO.
- #define **OVERSAMPLE_RATE** FAST_LOOP_HZ/FUSION_HZ

5.44.1 Detailed Description

A "standard" build configuration file.

This file contains only those parameters that directly relate to fusion implementation choices. Board dependencies are in hal_<shield_board>.h. Consult the Sensor Fusion User Guide for guidance and details.

5.45 status.c File Reference

Application-specific status subsystem.

```
#include "board.h"
#include "fsl_port.h"
#include "anomaly_detection.h"
#include "drivers.h"
#include "status.h"
```

Macros

- #define **N** 0x00
- #define **R** 0x04
- #define **G** 0x02
- #define **B** 0x01

Functions

- void **ssSetLeds** (int8_t RGB)
- void **ssSetStatusNow** (StatusSubsystem ∗pStatus, ad_status_t status)
- void **ssTest** (StatusSubsystem ∗pStatus)
- void **ssQueueStatus** (StatusSubsystem ∗pStatus, ad_status_t status)
- void **ssUpdateStatus** (StatusSubsystem ∗pStatus)
- void **ssSetStatus** (StatusSubsystem ∗pStatus, ad_status_t status)
- void initializeStatusSubsystem (StatusSubsystem ∗pStatus)

5.45.1 Detailed Description

Application-specific status subsystem.

Applications may change how they choose to display status information. The default implementation here uses LEDs on NXP Freedom boards. You may swap out implementations as long as the "Required" methods and states are retained.

5.45.2 Function Documentation

5.45.2.1 initializeStatusSubsystem()

```
void initializeStatusSubsystem (
            StatusSubsystem * pStatus )
```

initializeStatusSubsystem() should be called once at startup to initialize the data structure and to put hardware into the proper state for communicating status.

Parameters

pStatus	pointer to the status subsystem

Definition at line 189 of file status.c.

5.46 status.h File Reference

Application-specific status subsystem.

Data Structures

- struct StatusSubsystem
  StatusSubsystem() provides an object-like interface for communicating status to the user.

Typedefs

- typedef struct StatusSubsystem StatusSubsystem
  StatusSubsystem() provides an object-like interface for communicating status to the user.

Functions

- void initializeStatusSubsystem (StatusSubsystem ∗pStatus)
- void **ssSetStatusNow** (StatusSubsystem ∗pStatus, ad_status_t status)

5.46.1 Detailed Description

Application-specific status subsystem.

Applications may change how they choose to display status information. The default implementation here uses LEDs on NXP Freedom boards. You may swap out implementations as long as the "Required" methods and states are retained.

5.46.2 Function Documentation

5.46.2.1 initializeStatusSubsystem()

```
void initializeStatusSubsystem (
            StatusSubsystem * pStatus )
```

initializeStatusSubsystem() should be called once at startup to initialize the data structure and to put hardware into the proper state for communicating status.

Parameters

pStatus	pointer to the status subsystem

Definition at line 189 of file status.c.

Index