InvenSense Inc.
1197 Borregas Ave., Sunnyvale, CA 94089 U.S.A.
Tel: +1 (408) 988-7339 Fax: +1 (408) 988-8104
Website: www.invensense.com

Document Number :MOTAPPS-FS-V4.1.1
Release Date :10/21/2011

MPL Functional Specification Version 4.1.1

A printed copy of this document is
NOT UNDER REVISION CONTROL
unless it is dated and stamped in red ink as,
"REVISION CONTROLLED COPY."

Chapter 1

Purpose and Scope

This document is a guide to all of the functions available in the InvenSense Motion Processing Library (MPL), and corresponds with MPL Release v4.1.1. This release is designed to work with all MPU devices revision K or earlier.

MPL contains the code for controlling the InvenSense MPU series gyroscopes, including activating and managing built in motion processing features. All of the source code is in ANSI C and can be compiled in C or C++ environments.

All functions available in the MPL are described in this document, including all parameters involved in the function calls. The functions are divided into modules as follows:

Module	Name	Description
MLDMP	Motion Library DMP	Top level functions that define how to load the MPL.
ML	Motion Library	Controls basic operation of motion processing.
MLDL	ML Driver Layer	Used to configure hardware and low level ML functionality.
FIFO	Abstracted FIFO Driver Layer	Driver for the FIFO.
FIFOHW	Hardware FIFO Driver Layer	Driver for the HW FIFO.
ML_SUPERVISOR	ML Supervisor	Sensor Fusion supervisor.
COMPASSDL	Compass Driver Layer	Driver Layer for Compass support.
ACCELDL	Compass Driver Layer	Driver Layer for Compass support.
MLSL	ML System Layer	Hardware specific functions used by MLDL that must be written by the customer.
MPU_SELF_TEST	MPU Self Test	API to manage and trigger the run of the Self Test for gyros and accelerometers.
ML_STORED_DATA	ML Stored Data	Load and Store calibration APIs.
CONTROL	Control	Processes gyroscopes and accelerometers to provide control signals that can be used in user interfaces to manipulate objects such as documents, images, cursors, menus, etc.
PLUGIN_GESTURE	Gesture	Processes gyroscopes and accelerometers to provide recognition of a set of gestures.
PLUGIN_ORIENTATION	Orientation	Determines the orientation of device in the space.
PLUGIN_PEDOMETER_STAND_ALONE	Stand Alone Pedometer	Enables the step counting feature only.
PLUGIN_GLYPH	Glyphs	Character recognition engine.

For more information on how to use these functions in a specific application, refer to InvenSense Application Notes.

Chapter 2

About this document

This document is automatically generated from the source files using Doxygen's output format in the LaTeX. Heading, footer, and general document format are customized from the standard header template provided by Doxygen. This document is subdivided in the various sections, each describing the main source Modules composing the MPL and implementing specific features (e.g. Pedometer, Gesture Recognition Engine, etc...).

Every section starts with a brief description and an overview of the functions composing the module. Each of those functions is also fully documented in the analogous "Function Documentation" section. Clicking on the function prototype will lead to the portion of text full documentating it.

This **MPL Functional Specification** is best viewed in a PDF viewer, as it provides text hyperlinks and bookmarks on the left-hand side for ease of browsing. There is an Alphabatical Index of the modules and their functions available at the bottom of this document.

Chapter 3

Module Index

3.1 Modules

Here is a list of all modules:

2 Module Index

Chapter 4

Class Index

4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

4 Class Index

Chapter 5

Module Documentation

5.1 MLDMP

These are the top level functions that define how to load the MPL.

Files

- file mldmp.c
  Shared functions between all the different DMP versions.
- file mldmp.h
  Top level entry functions to the MPL library with DMP support.

Functions

- inv_error_t inv_dmp_close (void)
  Closes the motion sensor engine.
- inv_error_t inv_dmp_open (void)
  Open the default motion sensor engine.
- inv_error_t inv_dmp_start (void)
  Start the DMP.
- inv_error_t inv_dmp_stop (void)
  Stops the DMP and puts it in low power.

5.1.1 Detailed Description

These are the top level functions that define how to load the MPL.

In order to use most of the features, the DMP must be loaded with some code. The loading procedure takes place when calling inv_dmp_open with a given DMP set function, after having open the serial communication with the device via inv_serial_start(). The DMP set function will load the DMP memory and enable a certain set of features.

First select a DMP version from one of the released DMP sets. These could be:

- DMP default to load and use the default DMP code featuring pedometer, gestures, and orientation. Use inv_dmp_open().
- DMP pedometer stand-alone to load and use the standalone pedometer implementation. Use inv_open_low_power_pedometer().

After inv_dmp_openXXX any number of appropriate initialization and configuration routines can be called. Each one of these routines will return an error code and will check to make sure that it is compatible with the the DMP version selected during the call to inv_dmp_open.

Once the configuration is complete, make a call to inv_dmp_start(). This will finally turn on the DMP and run the code previously loaded.

While the DMP is running, all data fetching, polling or other functions can be called and will return valid data. Some parameteres can be changed while the DMP is running, while others cannot. Therefore it is important to always check the return code of each function. Check the error code list in mltypes to know what each returned error corresponds to.

When no more motion processing is required, the library can be shut down and the DMP turned off. We can do that by calling inv_dmp_close(). Note that inv_dmp_close() will not close the serial communication automatically, which will remain open an active, in case another module needs to be loaded instead. If the intention is shutting down the MPL as well, an explicit call to inv_serial_stop() following inv_dmp_close() has to be made.

The MPL additionally implements a basic state machine, whose purpose is to give feedback to the user on whether he is following all the required initialization steps. If an anomalous transition is detected, the user will be warned by a terminal message with the format:

```
"Error :  illegal state transition from STATE_1 to
STATE_3"
```

5.1.2 Function Documentation

5.1.2.1 inv_error_t inv_dmp_close (void)

Closes the motion sensor engine.

Does not close the serial communication. To do that, call inv_serial_stop(). After calling inv_dmp_close() another DMP module can be loaded in the MPL with the corresponding necessary intialization and configurations, via any of the MLDmpXXXOpen functions.

Precondition:

inv_dmp_open() must have been called.

```
result = inv_dmp_close();
if (INV_SUCCESS != result) {
    // Handle the error case
}
```

Returns:

INV_SUCCESS, Non-zero error code otherwise.

5.1.2.2 inv_error_t inv_dmp_open (void)

Open the default motion sensor engine.

This function is used to open the default MPL engine, featuring, for example, sensor fusion (6 axes and 9 axes), sensor calibration, accelerometer data byte swapping, among others. Compare with the other provided engines.

Precondition:

inv_serial_start() must have been called to instantiate the serial communication.

Example:

```
result = inv_dmp_open( );
if (INV_SUCCESS != result) {
    // Handle the error case
}
```

Returns:

Zero on success; Error Code on any failure.

5.1.2.3 inv_error_t inv_dmp_start (void)

Start the DMP.

Precondition:

inv_dmp_open() must have been called.

```
result = inv_dmp_start();
if (INV_SUCCESS != result) {
    // Handle the error case
}
```

Returns:

INV_SUCCESS if successful, or Non-zero error code otherwise.

5.1.2.4 inv_error_t inv_dmp_stop (void)

Stops the DMP and puts it in low power.

Precondition:

inv_dmp_start() must have been called.

Returns:

INV_SUCCESS, Non-zero error code otherwise.

5.2 ML

Motion Library APIs.

Files

- file ml.c

  The Motion Library APIs.

Functions

- inv_error_t inv_apply_calibration (void)

  apply the choosen orientation and full scale range for gyroscopes, accelerometer, and compass.

- inv_error_t inv_apply_endian_accel (void)

  Setup the DMP to handle the accelerometer endianess.

- int inv_check_flag (int flag)

  inv_check_flag returns the value of a flag.

- inv_error_t inv_disable_bias_from_gravity (void)

  inv_disable_bias_from_gravity *turns off the algorithm to produce gyro data from the 6-axis quaternion.*

- inv_error_t inv_disable_bias_from_LPF (void)

  inv_disable_bias_from_LPF *disables the algorithm to calculate gyroscope bias from LPF.*

- inv_error_t inv_enable_bias_from_gravity (int check_compass)

  inv_enable_bias_from_gravity *turns on the algorithm to produce gyro data from the 6-axis quaternion.*

- inv_error_t inv_enable_bias_from_LPF (int check_compass)

  inv_enable_bias_from_LPF *enables the algorithm to calculate gyroscope bias from LPF.*

- int inv_get_gyro_present (void)

  Check for the presence of the gyro sensor.

- int inv_get_interrupts (void)

Get the current set of DMP interrupt sources.

- int inv_get_motion_state (void)

  inv_get_motion_state is used to determine if the device is in a 'motion' or 'no motion' state.

- void * inv_get_serial_handle (void)

  Get the serial file handle to the device.

- inv_error_t inv_get_version (unsigned char **version)

  inv_get_version is used to get the ML version.

- inv_error_t inv_serial_start (char const *port)

  Open serial connection with the MPU device.

- inv_error_t inv_serial_stop (void)

  Close the serial communication.

- inv_error_t inv_set_compass_calibration (float range, signed char *orientation)

  Sets up the Compass calibration and scale factor.

- inv_error_t inv_set_dead_zone_high (void)

  inv_set_dead_zone_high *is used to set a large gyro dead zone.*

- inv_error_t inv_set_dead_zone_normal (int check_compass)

  inv_set_dead_zone_normal *is used to enable the gyro dead zone.*

- inv_error_t inv_set_dead_zone_zero (void)

  inv_set_dead_zone_zero *is used to disable the gyro dead zone.*

- inv_error_t inv_set_dmp_dr_interrupt (unsigned char on)

  Enable generation of the DMP interrupt when data is ready for the DMP.

- inv_error_t inv_set_fifo_interrupt (unsigned char on)

  Enable generation of the DMP interrupt when a FIFO packet is ready.

- inv_error_t inv_set_motion_interrupt (unsigned char on)

  Enable generation of the DMP interrupt when Motion or no-motion is detected.

- inv_error_t inv_set_mpu_sensors (unsigned long sensors)

  Controlls each sensor and each axis when the motion processing unit is running.

- inv_error_t inv_set_no_motion_thresh (float thresh)

  inv_set_no_motion_thresh is used to set the threshold for detecting INV_NO_MOTION

- inv_error_t inv_set_no_motion_threshAccel (long thresh)

  inv_set_no_motion_threshAccel is used to set the threshold for detecting INV_NO_MOTION with accelerometers when Gyros have been turned off

- inv_error_t inv_set_no_motion_time (float time)

  inv_set_no_motion_time is used to set the time required for detecting INV_NO_MOTION

- inv_error_t inv_update_data (void)

  inv_update_data fetches data from the fifo and updates the motion algorithms.

5.2.1 Detailed Description

Motion Library APIs.

The Motion Library processes gyroscopes, accelerometers, and compasses to provide a physical model of the movement for the sensors. The results of this processing may be used to control objects within a user interface environment, detect gestures, track 3D movement for gaming applications, and analyze the blur created due to hand movement while taking a picture.

5.2.2 Function Documentation

5.2.2.1 inv_error_t inv_apply_calibration (void)

apply the choosen orientation and full scale range for gyroscopes, accelerometer, and compass.

Returns:

INV_SUCCESS if successful, a non-zero code otherwise.

5.2.2.2 inv_error_t inv_apply_endian_accel (void)

Setup the DMP to handle the accelerometer endianess.

Returns:

INV_SUCCESS if successful, a non-zero error code otherwise.

5.2.2.3 int inv_check_flag (int *flag*)

inv_check_flag returns the value of a flag.

inv_check_flag can be used to check a number of flags, allowing users to poll flags rather than register callback functions. If a flag is set to True when inv_check_flag is called, the flag is automatically reset. The flags are:

- INV_RAW_DATA_READY Indicates that new raw data is available.
- INV_PROCESSED_DATA_READY Indicates that new processed data is available.
- INV_GOT_GESTURE Indicates that a gesture has been detected by the gesture engine.
- INV_MOTION_STATE_CHANGE Indicates that a change has been made from motion to no motion, or vice versa.

Precondition:

inv_dmp_open()

or inv_open_low_power_pedometer() or inv_eis_open_dmp()

and inv_dmp_start() must have been called.

Parameters:

flag The flag to check.

Returns:

true or false state of the flag

5.2.2.4 inv_error_t inv_disable_bias_from_gravity (void)

inv_disable_bias_from_gravity turns off the algorithm to produce gyro data from the 6-axis quaternion.

Returns:

INV_SUCCESS if successful.

5.2.2.5 inv_error_t inv_disable_bias_from_LPF (void)

inv_disable_bias_from_LPF disables the algorithm to calculate gyroscope bias from LPF.

Returns:

INV_SUCCESS if successful.

5.2.2.6 inv_error_t inv_enable_bias_from_gravity (int *check_compass*)

inv_enable_bias_from_gravity turns on the algorithm to produce gyro data from the 6-axis quaternion.

This function determines which type of data (raw gyro or accel-compensated gyro) will appear in the FIFO.

Parameters:

check_compass If 1, algorithm is only used when compass is not used.

Returns:

INV_SUCCESS if successful.

5.2.2.7 inv_error_t inv_enable_bias_from_LPF (int *check_compass*)

inv_enable_bias_from_LPF enables the algorithm to calculate gyroscope bias from LPF.

Parameters:

check_compass If 1, only update bias from LPF if compass is not present.

Returns:

INV_SUCCESS if successful.

5.2.2.8 int inv_get_gyro_present (void)

Check for the presence of the gyro sensor.

This is not a physical check but a logical check and the value can change dynamically based on calls to inv_set_mpu_sensors().

Returns:

true if the gyro is enabled false otherwise.

5.2.2.9 int inv_get_interrupts (void)

Get the current set of DMP interrupt sources.

These interrupts are generated by the DMP and can be routed to the MPU interrupt line via internal settings.

Precondition:

inv_dmp_open()

or inv_open_low_power_pedometer() or inv_eis_open_dmp()

must have been called.

Returns:

Currently enabled interrupt sources. The possible interrupts are:

- INV_INT_FIFO,
- INV_INT_MOTION,
- INV_INT_TAP

5.2.2.10 int inv_get_motion_state (void)

inv_get_motion_state is used to determine if the device is in a 'motion' or 'no motion' state.

inv_get_motion_state returns INV_MOTION of the device is moving, or INV_NO_-MOTION if the device is not moving.

Precondition:

inv_dmp_open()

or inv_open_low_power_pedometer() or inv_eis_open_dmp()

and inv_dmp_start() must have been called.

Returns:

INV_SUCCESS if successful or Non-zero error code otherwise.

5.2.2.11 void* inv_get_serial_handle (void)

Get the serial file handle to the device.

Returns:

The serial file handle.

5.2.2.12 inv_error_t inv_get_version (unsigned char ** *version*)

inv_get_version is used to get the ML version.

Precondition:

inv_get_version can be called at any time.

Parameters:

version inv_get_version writes the ML version string pointer to version.

Returns:

INV_SUCCESS if successful or Non-zero error code otherwise.

5.2.2.13 inv_error_t inv_serial_start (char const * *port*)

Open serial connection with the MPU device.

This is the entry point of the MPL and must be called prior to any other function call.

Parameters:

port System handle for 'port' MPU device is found on. The significance of this parameter varies by platform. It is passed as 'port' to MLSLSerialOpen.

Returns:

INV_SUCCESS or error code.

5.2.2.14 inv_error_t inv_serial_stop (void)

Close the serial communication.

This function needs to be called explicitly to shut down the communication with the device. Calling inv_dmp_close() won't affect the established serial communication.

Returns:

INV_SUCCESS; non-zero error code otherwise.

5.2.2.15 inv_error_t inv_set_compass_calibration (float *range*, signed char * *orientation*)

Sets up the Compass calibration and scale factor.

Please refer to the provided "9-Axis Sensor Fusion Application Note" document provided. Section 5, "Sensor Mounting Orientation" offers a good coverage on the mounting matrices and explains how to use them.

Precondition:

inv_dmp_open()

or inv_open_low_power_pedometer() or inv_eis_open_dmp()

must have been called.

Precondition:

inv_dmp_start() must have **NOT** been called.

See also:

inv_set_gyro_calibration().
inv_set_accel_calibration().

Parameters:

range The range of the compass.

orientation A 9 element matrix that represents how the compass is oriented with respect to the device they are mounted in. A typical set of values are {1, 0, 0, 0, 1, 0, 0, 0, 1}. This example corresponds to a 3 x 3 identity matrix. The matrix describes how to go from the chip mounting to the body of the device.

Returns:

INV_SUCCESS if successful or Non-zero error code otherwise.

5.2.2.16 inv_error_t inv_set_dead_zone_high (void)

inv_set_dead_zone_high is used to set a large gyro dead zone.

This setting is typically used when high amounts of jitter are expected.For the 3050, calibrated gyro data can be zero as a result. For 6050, only the quaternion will be affected.

Returns:

INV_SUCCESS if successful.

5.2.2.17 inv_error_t inv_set_dead_zone_normal (int *check_compass*)

inv_set_dead_zone_normal is used to enable the gyro dead zone.

This setting can be configured such that the dead zone is only enabled when the compass is not used. For the 3050, calibrated gyro data can be zero as a result. For 6050, only the quaternion will be affected.

Parameters:

check_compass If 1, only enable dead zone if compass is not present.

Returns:

INV_SUCCESS if successful.

5.2.2.18 inv_error_t inv_set_dead_zone_zero (void)

inv_set_dead_zone_zero is used to disable the gyro dead zone.

Returns:

INV_SUCCESS if successful.

5.2.2.19 inv_error_t inv_set_dmp_dr_interrupt (unsigned char *on*)

Enable generation of the DMP interrupt when data is ready for the DMP.

This IRQ can be used to get a timestamp right before the DMP starts processing the data. The FIFO interrupt can be between 2 and 5 ms later.

Parameters:

on Boolean to turn the interrupt on or off

Returns:

INV_SUCCESS or non-zero error code

5.2.2.20 inv_error_t inv_set_fifo_interrupt (unsigned char *on*)

Enable generation of the DMP interrupt when a FIFO packet is ready.

Parameters:

on Boolean to turn the interrupt on or off

Returns:

INV_SUCCESS or non-zero error code

5.2.2.21 inv_error_t inv_set_motion_interrupt (unsigned char *on*)

Enable generation of the DMP interrupt when Motion or no-motion is detected.

Parameters:

on Boolean to turn the interrupt on or off.

Returns:

INV_SUCCESS or non-zero error code.

5.2.2.22 inv_error_t inv_set_mpu_sensors (unsigned long *sensors*)

Controlls each sensor and each axis when the motion processing unit is running.

When it is not running, simply records the state for later.

NOTE: In this version only full sensors controll is allowed. Independent axis control will return an error.

Parameters:

sensors Bit field of each axis desired to be turned on or off

Returns:

INV_SUCCESS or non-zero error code

5.2.2.23 inv_error_t inv_set_no_motion_thresh (float *thresh*)

inv_set_no_motion_thresh is used to set the threshold for detecting INV_NO_-MOTION

Precondition:

inv_dmp_open()

or inv_open_low_power_pedometer() or inv_eis_open_dmp()

must have been called.

Parameters:

thresh A threshold scaled in degrees per second.

Returns:

INV_SUCCESS if successful or Non-zero error code otherwise.

5.2.2.24 inv_error_t inv_set_no_motion_threshAccel (long *thresh*)

inv_set_no_motion_threshAccel is used to set the threshold for detecting INV_NO_-MOTION with accelerometers when Gyros have been turned off

Precondition:

inv_dmp_open()

or inv_open_low_power_pedometer() or inv_eis_open_dmp()

must have been called.

Parameters:

thresh A threshold in g's scaled by 2^32

Returns:

INV_SUCCESS if successful or Non-zero error code otherwise.

5.2.2.25 inv_error_t inv_set_no_motion_time (float *time*)

inv_set_no_motion_time is used to set the time required for detecting INV_NO_-MOTION

Precondition:

inv_dmp_open()

or inv_open_low_power_pedometer() or inv_eis_open_dmp()

must have been called.

Parameters:

time A time in seconds.

Returns:

INV_SUCCESS if successful or Non-zero error code otherwise.

5.2.2.26 inv_error_t inv_update_data (void)

inv_update_data fetches data from the fifo and updates the motion algorithms.

Precondition:

inv_dmp_open()

or inv_open_low_power_pedometer() or inv_eis_open_dmp()

and inv_dmp_start() must have been called.

Note:

Motion algorithm data is constant between calls to inv_update_data

Returns:

- INV_SUCCESS
- Non-zero error code

5.3 MLARRAY

Files

- file mlarray_adv.c

  APIs to read different data sets from FIFO.

Functions

- inv_error_t inv_get_compass_accuracy (int *accuracy)

  Returns the curren compass accuracy.

- inv_error_t inv_get_local_field (long *data)

  inv_get_local_field is used to get local magnetic field data.

- inv_error_t inv_get_local_field_float (float *data)

  inv_get_local_field_float is used to get local magnetic field data.

- inv_error_t inv_get_mag_bias_error (long *data)

  inv_get_mag_bias_error is used to get magnetometer Bias error.

- inv_error_t inv_get_mag_bias_error_float (float *data)

  inv_get_mag_bias_error_float is used to get an array of three numbers representing the current estimated error in the compass biases.

- inv_error_t inv_get_mag_scale (long *data)

  inv_get_mag_scale is used to get magnetometer scale.

- inv_error_t inv_get_mag_scale_float (float *data)

  inv_get_mag_scale_float is used to get magnetometer scale.

- inv_error_t inv_set_local_field (long *data)

  inv_set_local_field is used to set local magnetic field

- inv_error_t inv_set_local_field_float (float *data)

  inv_set_local_field_float is used to set local magnetic field

- inv_error_t inv_set_mag_scale (long *data)

  inv_set_mag_scale is used to set magnetometer scale

- inv_error_t inv_set_mag_scale_float (float *data)

  inv_set_mag_scale_float is used to set magnetometer scale

5.3.1 Function Documentation

5.3.1.1 inv_error_t inv_get_compass_accuracy (int * *accuracy*)

Returns the curren compass accuracy.

- 0: Unknown: The accuracy is unreliable and compass data should not be used
- 1: Low: The compass accuracy is low.
- 2: Medium: The compass accuracy is medium.
- 3: High: The compas acurracy is high and can be trusted

Parameters:

accuracy The accuracy level in the range 0-3

Returns:

ML_SUCCESS or non-zero error code

5.3.1.2 inv_error_t inv_get_local_field (long * *data*)

inv_get_local_field is used to get local magnetic field data.

Precondition:

MLDmpOpen() or MLDmpPedometerStandAloneOpen() must have been called.

Parameters:

data A pointer to an array to be passed back to the user. **Must be 3 cells long at least**.

Returns:

Zero if the command is successful; an ML error code otherwise.

5.3.1.3 inv_error_t inv_get_local_field_float (float * *data*)

inv_get_local_field_float is used to get local magnetic field data.

Precondition:

MLDmpOpen() or MLDmpPedometerStandAloneOpen() must have been called.

Parameters:

data A pointer to an array to be passed back to the user. **Must be 3 cells long**.

Returns:

INV_SUCCESS if the command is successful; an error code otherwise.

5.3.1.4 inv_error_t inv_get_mag_bias_error (long * *data*)

inv_get_mag_bias_error is used to get magnetometer Bias error.

Precondition:

MLDmpOpen() or MLDmpPedometerStandAloneOpen() must have been called.

Parameters:

data A pointer to an array to be passed back to the user. **Must be 3 cells long at least**.

Returns:

Zero if the command is successful; an ML error code otherwise.

5.3.1.5 inv_error_t inv_get_mag_bias_error_float (float * *data*)

inv_get_mag_bias_error_float is used to get an array of three numbers representing the current estimated error in the compass biases.

These numbers are unitless and serve as rough estimates in which numbers less than 100 typically represent reasonably well calibrated compass axes.

Precondition:

MLDmpOpen() or MLDmpPedometerStandAloneOpen() must have been called.

Parameters:

data A pointer to an array to be passed back to the user. **Must be 3 cells long**.

Returns:

INV_SUCCESS if the command is successful; an error code otherwise.

5.3.1.6 inv_error_t inv_get_mag_scale (long * *data*)

inv_get_mag_scale is used to get magnetometer scale.

Precondition:

MLDmpOpen() or MLDmpPedometerStandAloneOpen() must have been called.

Parameters:

data A pointer to an array to be passed back to the user. **Must be 3 cells long at least**.

Returns:

Zero if the command is successful; an ML error code otherwise.

5.3.1.7 inv_error_t inv_get_mag_scale_float (float * *data*)

inv_get_mag_scale_float is used to get magnetometer scale.

Precondition:

MLDmpOpen() or MLDmpPedometerStandAloneOpen() must have been called.

Parameters:

data A pointer to an array to be passed back to the user. **Must be 3 cells long**.

Returns:

INV_SUCCESS if the command is successful; an error code otherwise.

5.3.1.8 inv_error_t inv_set_local_field (long * *data*)

inv_set_local_field is used to set local magnetic field

Please refer to the provided "9-Axis Sensor Fusion Application Note" document provided.

Precondition:

MLDmpOpen() or MLDmpPedometerStandAloneOpen()
MLDmpStart() must **NOT** have been called.

Parameters:

data A pointer to an array to be copied from the user.

Returns:

INV_SUCCESS if successful; a non-zero error code otherwise.

5.3.1.9 inv_error_t inv_set_local_field_float (float * *data*)

inv_set_local_field_float is used to set local magnetic field

Please refer to the provided "9-Axis Sensor Fusion Application Note" document provided.

Precondition:

MLDmpOpen() or MLDmpPedometerStandAloneOpen()
MLDmpStart() must **NOT** have been called.

Parameters:

data A pointer to an array to be copied from the user.

Returns:

INV_SUCCESS if successful; a non-zero error code otherwise.

5.3.1.10 inv_error_t inv_set_mag_scale (long * *data*)

inv_set_mag_scale is used to set magnetometer scale

Please refer to the provided "9-Axis Sensor Fusion Application Note" document provided.

Precondition:

MLDmpOpen() or MLDmpPedometerStandAloneOpen()
MLDmpStart() must **NOT** have been called.

Parameters:

data A pointer to an array to be copied from the user.

Returns:

INV_SUCCESS if successful; a non-zero error code otherwise.

5.3.1.11 inv_error_t inv_set_mag_scale_float (float * *data*)

inv_set_mag_scale_float is used to set magnetometer scale

Please refer to the provided "9-Axis Sensor Fusion Application Note" document provided.

Precondition:

MLDmpOpen() or MLDmpPedometerStandAloneOpen()
MLDmpStart() must **NOT** have been called.

Parameters:

data A pointer to an array to be copied from the user.

Returns:

INV_SUCCESS if successful; a non-zero error code otherwise.

5.4 MLARRAY_LEGACY

Legacy Motion Library Array APIs.

Files

- file mlarray_legacy.c

  The Legacy Motion Library Array APIs.

Functions

- inv_error_t inv_get_array (int dataSet, long ∗data)

  inv_get_array is used to get an array of processed motion sensor data.

- inv_error_t inv_get_float_array (int dataSet, float ∗data)

  inv_get_float_array is used to get an array of processed motion sensor data.

- inv_error_t inv_set_array (int dataSet, long ∗data)

  used to set an array of motion sensor data.

- inv_error_t inv_set_float_array (int dataSet, float ∗data)

  used to set an array of motion sensor data.

5.4.1 Detailed Description

Legacy Motion Library Array APIs.

The Motion Library Array APIs provide the user access to the Motion Library state. These Legacy APIs provide access to individual state arrays using a data set name as the first argument to the API. This format has been replaced by unique named APIs for each data set, found in the MLArray group.

5.4.2 Function Documentation

5.4.2.1 inv_error_t inv_get_array (int *dataSet*, long ∗ *data*)

inv_get_array is used to get an array of processed motion sensor data.

inv_get_array can be used to retrieve various data sets. Certain data sets require functions to be enabled using MLEnable in order to be valid.

The available data sets are:

- INV_ROTATION_MATRIX
- INV_QUATERNION
- INV_EULER_ANGLES_X
- INV_EULER_ANGLES_Y
- INV_EULER_ANGLES_Z
- INV_EULER_ANGLES
- INV_LINEAR_ACCELERATION
- INV_LINEAR_ACCELERATION_WORLD
- INV_GRAVITY
- INV_ANGULAR_VELOCITY
- INV_RAW_DATA
- INV_GYROS
- INV_ACCELS
- INV_MAGNETOMETER
- INV_GYRO_BIAS
- INV_ACCEL_BIAS
- INV_MAG_BIAS
- INV_HEADING
- INV_MAG_BIAS_ERROR
- INV_PRESSURE

Please refer to the documentation of inv_get_float_array() for a description of these data sets.

Precondition:

MLDmpOpen() or MLDmpPedometerStandAloneOpen() must have been called.

Parameters:

dataSet A constant specifying an array of data processed by the motion processor.

data A pointer to an array to be passed back to the user. **Must be 9 cells long at least**.

Returns:

Zero if the command is successful; an ML error code otherwise.

5.4.2.2 inv_error_t inv_get_float_array (int *dataSet*, float * *data*)

inv_get_float_array is used to get an array of processed motion sensor data.

inv_get_array can be used to retrieve various data sets. Certain data sets require functions to be enabled using MLEnable in order to be valid.

The available data sets are:

- INV_ROTATION_MATRIX : Returns an array of nine data points representing the rotation matrix generated from all available sensors. This requires that ML_SENSOR_FUSION be enabled. The array format will be R11, R12, R13, R21, R22, R23, R31, R32, R33, representing the matrix:

  R11 R12 R13

  R21 R22 R23

  R31 R32 R33

  Please refer to the "9-Axis Sensor Fusion Application Note" document, section 7 "Sensor Fusion Output", for details regarding rotation matrix output.

- INV_QUATERNION : Returns an array of four data points representing the quaternion generated from all available sensors. This requires that ML_SENSOR_FUSION be enabled.

- INV_EULER_ANGLES_X : Returns an array of three data points representing roll, pitch, and yaw using the X axis of the gyroscope, accelerometer, and compass as reference axis. This is typically the convention used for mobile devices where the X axis is the width of the screen, Y axis is the height, and Z the depth. In this case roll is defined as the rotation around the X axis of the device. The euler angles convention for this output is the following:

EULER ANGLE	ROTATION AROUND
roll	X axis
pitch	Y axis
yaw	Z axis

  INV_EULER_ANGLES_X corresponds to the INV_EULER_ANGLES output and is therefore the default convention.

- INV_EULER_ANGLES_Y : Returns an array of three data points representing roll, pitch, and yaw using the Y axis of the gyroscope, accelerometer, and compass as reference axis. This convention is typically used in augmented reality applications, where roll is defined as the rotation around the axis along the height of the screen of a mobile device, namely the Y axis. The euler angles convention for this output is the following:

EULER ANGLE	ROTATION AROUND
roll	Y axis
pitch	X axis
yaw	Z axis

- INV_EULER_ANGLES_Z : Returns an array of three data points representing roll, pitch, and yaw using the Z axis of the gyroscope, accelerometer, and compass as reference axis. This convention is mostly used in application involving the use of a camera, typically placed on the back of a mobile device, that is along the Z axis. In this convention roll is defined as the rotation around the Z axis. The euler angles convention for this output is the following:

EULER ANGLE	ROTATION AROUND
roll	Z axis
pitch	X axis
yaw	Y axis

- INV_EULER_ANGLES : Returns an array of three data points representing roll, pitch, and yaw corresponding to the INV_EULER_ANGLES_X output and it is therefore the default convention for Euler angles. Please refer to the INV_-EULER_ANGLES_X for a detailed description.

- INV_LINEAR_ACCELERATION : Returns an array of three data points representing the linear acceleration as derived from both gyroscopes and accelerometers. This requires that ML_SENSOR_FUSION be enabled.

- INV_LINEAR_ACCELERATION_WORLD : Returns an array of three data points representing the linear acceleration in world coordinates, as derived from both gyroscopes and accelerometers. This requires that ML_SENSOR_FUSION be enabled.

- INV_GRAVITY : Returns an array of three data points representing the direction of gravity in body coordinates, as derived from both gyroscopes and accelerometers. This requires that ML_SENSOR_FUSION be enabled.

- INV_ANGULAR_VELOCITY : Returns an array of three data points representing the angular velocity as derived from **both** gyroscopes and accelerometers. This requires that ML_SENSOR_FUSION be enabled, to fuse data from the gyroscope and accelerometer device, appropriately scaled and oriented according to the respective mounting matrices.

- INV_RAW_DATA : Returns an array of nine data points representing raw sensor data of the gyroscope X, Y, Z, accelerometer X, Y, Z, and compass X, Y, Z values. These values are not scaled and come out directly from the devices' sensor data output. In case of accelerometers with lower output resolution, e.g 8-bit, the sensor data is scaled up to match the 2^14 = 1 gee typical representation for a +/- 2 gee full scale range.

- INV_GYROS : Returns an array of three data points representing the X gyroscope, Y gyroscope, and Z gyroscope values. The values are not sensor fused with other sensor types data but reflect the orientation from the mounting matrices in use. The INV_GYROS values are scaled to ensure 1 dps corresponds to 2^16 codes.

- INV_ACCELS : Returns an array of three data points representing the X accelerometer, Y accelerometer, and Z accelerometer values. The values are not sensor fused with other sensor types data but reflect the orientation from the mounting matrices in use. The INV_ACCELS values are scaled to ensure 1 gee corresponds to 2^16 codes.

- INV_MAGNETOMETER : Returns an array of three data points representing the compass X, Y, and Z values. The values are not sensor fused with other sensor types data but reflect the orientation from the mounting matrices in use. The INV_MAGNETOMETER values are scaled to ensure 1 micro Tesla (uT) corresponds to 2^16 codes.

- INV_GYRO_BIAS : Returns an array of three data points representing the gyroscope biases.

- INV_ACCEL_BIAS : Returns an array of three data points representing the accelerometer biases.

- INV_MAG_BIAS : Returns an array of three data points representing the compass biases.

- INV_GYRO_CALIBRATION_MATRIX : Returns an array of nine data points representing the calibration matrix for the gyroscopes:

  C11 C12 C13

  C21 C22 C23

  C31 C32 C33

- INV_ACCEL_CALIBRATION_MATRIX : Returns an array of nine data points representing the calibration matrix for the accelerometers:

C11 C12 C13

C21 C22 C23

C31 C32 C33

- INV_MAG_CALIBRATION_MATRIX : Returns an array of nine data points representing the calibration matrix for the compass:

C11 C12 C13

C21 C22 C23

C31 C32 C33

- INV_PRESSURE : Returns a single value representing the pressure in Pascal

- INV_HEADING : Returns a single number representing the heading of the device relative to the Earth, in which 0 represents North, 90 degrees represents East, and so on. The heading is defined as the direction of the +Y axis if the Y axis is horizontal, and otherwise the direction of the -Z axis.

- INV_MAG_BIAS_ERROR : Returns an array of three numbers representing the current estimated error in the compass biases. These numbers are unitless and serve as rough estimates in which numbers less than 100 typically represent reasonably well calibrated compass axes.

Precondition:

MLDmpOpen() or MLDmpPedometerStandAloneOpen() must have been called.

Parameters:

dataSet A constant specifying an array of data processed by the motion processor.

data A pointer to an array to be passed back to the user. **Must be 9 cells long at least**.

Returns:

INV_SUCCESS if the command is successful; an error code otherwise.

5.4.2.3 inv_error_t inv_set_array (int *dataSet*, long * *data*)

used to set an array of motion sensor data.

Handles the following data sets:

- INV_GYRO_BIAS
- INV_ACCEL_BIAS
- INV_MAG_BIAS
- INV_GYRO_TEMP_SLOPE

For more details about the use of the data sets please refer to the documentation of inv_set_float_array().

Please also refer to the provided "9-Axis Sensor Fusion Application Note" document provided.

Precondition:

MLDmpOpen() or MLDmpPedometerStandAloneOpen()
MLDmpStart() must **NOT** have been called.

Parameters:

dataSet A constant specifying an array of data.

data A pointer to an array to be copied from the user.

Returns:

INV_SUCCESS if successful; a non-zero error code otherwise.

5.4.2.4 inv_error_t inv_set_float_array (int *dataSet*, float * *data*)

used to set an array of motion sensor data.

Handles various data sets:

- INV_GYRO_BIAS
- INV_ACCEL_BIAS
- INV_MAG_BIAS
- INV_GYRO_TEMP_SLOPE

Please refer to the provided "9-Axis Sensor Fusion Application Note" document provided.

Precondition:

MLDmpOpen() or MLDmpPedometerStandAloneOpen()
MLDmpStart() must **NOT** have been called.

Parameters:

dataSet A constant specifying an array of data.

data A pointer to an array to be copied from the user.

Returns:

INV_SUCCESS if successful; a non-zero error code otherwise.

5.5 MLFIFO

Motion Library - FIFO Driver.

Files

- file mlFIFO.c

  FIFO Interface.

Functions

- unsigned long inv_accel_sum_of_sqr (void)

  The gyro data magnitude squared: (1 g)^2 = 2^16 = 2^ACC_MAG_SQR_SHIFT.

- inv_error_t inv_check_fifo_callback (inv_obj_func callback, unsigned char *is_registered)

  checks if a FIFO callback has already been registered.

- inv_error_t inv_close_fifo (void)

  Close the FIFO usage.

- long inv_decode_temperature (short temp_reg)

  Converts 16-bit temperature data as read from temperature register into Celcius scaled by 2^16.

- inv_error_t inv_get_6axis_quaternion (long *data)

  Returns 4-element quaternion vector derived from 6 axis sensors (gyros and accels).

- inv_error_t inv_get_accel (long *data)

  Returns 3-element vector of accelerometer data in body frame.

- inv_error_t inv_get_accel_float (float *data)

  Returns 3-element vector of accelerometer data in body frame.

- inv_error_t inv_get_cntrl_data (long *data)

  Returns 4-element vector of control data.

- inv_error_t inv_get_eis (long *data)

  Returns 3-element vector of EIS shfit data.

- inv_error_t inv_get_external_sensor_data (long *data, int size)

Returns 3-element vector of external sensor.

- unsigned short inv_get_fifo_rate (void)

  Retrieve the current FIFO update divider - 1.

- inv_error_t inv_get_gravity (long *data)

  Get the 3-element gravity vector from the FIFO expressed in coordinates relative to the body frame.

- inv_error_t inv_get_gyro (long *data)

  Returns 3-element vector of gyro data in body frame.

- inv_error_t inv_get_gyro_and_accel_sensor (long *data)

  Returns 6-element vector of gyro and accel data.

- inv_error_t inv_get_gyro_raw (long *data)

  Returns raw gyro data in the body frame.

- inv_error_t inv_get_gyro_raw_float (float *data)

  Returns raw gyro data in the body frame.

- inv_error_t inv_get_gyro_sensor (long *data)

  This gets raw gyro data.

- unsigned long inv_get_gyro_sum_of_sqr (void)

  The gyro data magnitude squared : (1 degree per second)^2 = 2^6 = 2^GYRO_MAG_SQR_SHIFT.

- inv_error_t inv_get_linear_accel (long *data)

  Returns 3-element vector of accelerometer data in body frame with gravity removed.

- inv_error_t inv_get_linear_accel_in_world (long *data)

  Returns 3-element vector of accelerometer data in world frame with gravity removed.

- inv_error_t inv_get_packet_number (uint16_t *data)

  Returns value of packet number.

- inv_error_t inv_get_quantized_accel (long *data)

  Get the Quantized Accel data algorithm output from the FIFO.

- inv_error_t inv_get_quaternion (long *data)

  Returns 4-element quaternion vector derived from 6-axis or 9-axis if 9-axis was implemented.

- inv_error_t inv_get_quaternion_float (float *data)
  Returns 4-element quaternion vector.
- int_fast16_t inv_get_sample_frequency (void)
  Returns the step size for quaternion type data.
- int_fast16_t inv_get_sample_step_size_ms (void)
  Returns the step size for quaternion type data.
- inv_error_t inv_get_temperature (long *data)
  Returns 1-element vector of temperature.
- inv_error_t inv_get_unquantized_accel (long *data)
  Get the Decoded Accel Data.
- inv_error_t inv_init_fifo_param (void)
  Initializes all the internal static variables for the FIFO module.
- inv_error_t inv_read_and_process_fifo (int_fast8_t numPackets, int_fast8_t *processed)
  Reads and processes FIFO data.
- inv_error_t inv_send_accel (uint_fast16_t elements, uint_fast16_t accuracy)
  Sends accelerometer data to the FIFO.
- inv_error_t inv_send_cntrl_data (uint_fast16_t elements, uint_fast16_t accuracy)
  Sends control data to the FIFO.
- inv_error_t inv_send_external_sensor_data (uint_fast16_t elements, uint_fast16_t accuracy)
  Sends raw external data to the FIFO.
- inv_error_t inv_send_gravity (uint_fast16_t elements, uint_fast16_t accuracy)
  Send the computed gravity vectors into the FIFO.
- inv_error_t inv_send_gyro (uint_fast16_t elements, uint_fast16_t accuracy)
  Sends gyro data to the FIFO.
- inv_error_t inv_send_linear_accel (uint_fast16_t elements, uint_fast16_t accuracy)
  Sends linear accelerometer data to the FIFO.

- inv_error_t inv_send_linear_accel_in_world (uint_fast16_t elements, uint_fast16_t accuracy)
  Sends linear world accelerometer data to the FIFO.
- inv_error_t inv_send_packet_number (uint_fast16_t accuracy)
  Adds a rolling counter to the fifo packet.
- inv_error_t inv_send_quantized_accel (uint_fast16_t elements, uint_fast16_t accuracy)
  Send the Quantized Acceleromter data into the FIFO.
- inv_error_t inv_send_quaternion (uint_fast16_t accuracy)
  Sends quaternion data to the FIFO.
- inv_error_t inv_send_sensor_data (uint_fast16_t elements, uint_fast16_t accuracy)
  Sends raw data to the FIFO.
- inv_error_t inv_set_fifo_processed_callback (void(∗func)(void))
  inv_set_fifo_processed_callback is used to set a processed data callback function.
- inv_error_t inv_set_fifo_rate (unsigned short fifoRate)
  Command the MPU to put data in the FIFO at a particular rate.
- inv_error_t inv_set_gyro_data_source (uint_fast8_t source)
  Set the gyro source to output to the fifo.
- inv_error_t inv_set_linear_accel_filter_coef (float coef)
  Sets the filter coefficent used for computing the acceleration bias which is used to compute linear acceleration.

5.5.1 Detailed Description

Motion Library - FIFO Driver.

The FIFO API interface.

5.5.2 Function Documentation

5.5.2.1 unsigned long inv_accel_sum_of_sqr (void)

The gyro data magnitude squared: (1 g)^2 = 2^16 = 2^ACC_MAG_SQR_SHIFT.

Returns:

the computed magnitude squared output of the accelerometer.

5.5.2.2 inv_error_t inv_check_fifo_callback (inv_obj_func *callback*, unsigned char * *is_registered*)

checks if a FIFO callback has already been registered.

Parameters:

callback Callback function.

is_registered 1 if function is registered.

Returns:

INV_SUCCESS if successful.

5.5.2.3 inv_error_t inv_close_fifo (void)

Close the FIFO usage.

Returns:

INV_SUCCESS if successful, a non-zero error code otherwise.

5.5.2.4 long inv_decode_temperature (short *temp_reg*)

Converts 16-bit temperature data as read from temperature register into Celcius scaled by 2^16.

Parameters:

temp_reg the temperature expressed in the internal device representation as a 2-bytes word.

5.5.2.5 inv_error_t inv_get_6axis_quaternion (long * *data*)

Returns 4-element quaternion vector derived from 6 axis sensors (gyros and accels).

Parameters:

data 4-element quaternion vector. One is scaled to 2^30.

Returns:

0 on success or an error code.

5.5.2.6 inv_error_t inv_get_accel (long * *data*)

Returns 3-element vector of accelerometer data in body frame.

Parameters:

data 3-element vector of accelerometer data in body frame. One gee = 2^16.

Returns:

0 on success or an error code.

5.5.2.7 inv_error_t inv_get_accel_float (float * *data*)

Returns 3-element vector of accelerometer data in body frame.

Parameters:

data 3-element vector of accelerometer data in body frame in g's.

Returns:

0 for success or an error code.

5.5.2.8 inv_error_t inv_get_cntrl_data (long * *data*)

Returns 4-element vector of control data.

Parameters:

data 4-element vector of control data.

Returns:

0 for succes or an error code.

5.5.2.9 inv_error_t inv_get_eis (long * *data*)

Returns 3-element vector of EIS shfit data.

Parameters:

data 3-element vector of EIS shift data.

Returns:

0 for succes or an error code.

5.5.2.10 inv_error_t inv_get_external_sensor_data (long * *data*, int *size*)

Returns 3-element vector of external sensor.

Parameters:

data 3-element vector of external sensor

size the size of the buffer.

Returns:

0 on success or an error code.

5.5.2.11 unsigned short inv_get_fifo_rate (void)

Retrieve the current FIFO update divider - 1.

See inv_set_fifo_rate() for how this value is used.

The fifo rate when there is no fifo is the equivilent divider when derived from the value set by SetSampleSteSizeMs()

Returns:

The value of the fifo rate divider or INV_INVALID_FIFO_RATE on error.

5.5.2.12 inv_error_t inv_get_gravity (long * *data*)

Get the 3-element gravity vector from the FIFO expressed in coordinates relative to the body frame.

Parameters:

data 3-element vector of gravity in body frame.

Returns:

0 on success or an error code.

5.5.2.13 inv_error_t inv_get_gyro (long * *data*)

Returns 3-element vector of gyro data in body frame.

Parameters:

data 3-element vector of gyro data in body frame with gravity removed. One degree per second = 2^16.

Returns:

0 on success or an error code.

5.5.2.14 inv_error_t inv_get_gyro_and_accel_sensor (long * *data*)

Returns 6-element vector of gyro and accel data.

Parameters:

data 6-element vector of gyro and accel data

Returns:

0 on success or an error code.

5.5.2.15 inv_error_t inv_get_gyro_raw (long * *data*)

Returns raw gyro data in the body frame.

Parameters:

data Gyro data. 1 dps = 2^16.

Returns:

INV_SUCCESS if successful.

5.5.2.16 inv_error_t inv_get_gyro_raw_float (float * *data*)

Returns raw gyro data in the body frame.

Parameters:

data Gyro data.

Returns:

INV_SUCCESS if successful.

5.5.2.17 inv_error_t inv_get_gyro_sensor (long * *data*)

This gets raw gyro data.

The data is taken from the FIFO if it was put in the FIFO and it is read from the registers if it was not put into the FIFO. The data is cached till the next FIFO processing block time.

Parameters:

data Length 3, Gyro data

5.5.2.18 unsigned long inv_get_gyro_sum_of_sqr (void)

The gyro data magnitude squared : (1 degree per second)^2 = 2^6 = 2^GYRO_MAG_SQR_SHIFT.

Returns:

the computed magnitude squared output of the gyroscope.

5.5.2.19 inv_error_t inv_get_linear_accel (long * *data*)

Returns 3-element vector of accelerometer data in body frame with gravity removed.

Parameters:

data 3-element vector of accelerometer data in body frame with gravity removed. One g = 2^16.

Returns:

0 on success or an error code. data unchanged on error.

5.5.2.20 inv_error_t inv_get_linear_accel_in_world (long * *data*)

Returns 3-element vector of accelerometer data in world frame with gravity removed.

Parameters:

data 3-element vector of accelerometer data in world frame with gravity removed. One g = 2^16.

Returns:

0 on success or an error code.

5.5.2.21 inv_error_t inv_get_packet_number (uint16_t * *data*)

Returns value of packet number.

Parameters:

data 1-element vector of packet number

Returns:

0 for succes or an error code.

5.5.2.22 inv_error_t inv_get_quantized_accel (long * *data*)

Get the Quantized Accel data algorithm output from the FIFO.

Parameters:

data a buffer to store the quantized data.

Returns:

INV_SUCCESS if successful, a non-zero error code otherwise.

5.5.2.23 inv_error_t inv_get_quaternion (long * *data*)

Returns 4-element quaternion vector derived from 6-axis or 9-axis if 9-axis was implemented.

6-axis is gyros and accels. 9-axis is gyros, accel and compass.

Parameters:

data 4-element quaternion vector. One is scaled to 2^30.

Returns:

0 on success or an error code.

5.5.2.24 inv_error_t inv_get_quaternion_float (float * *data*)

Returns 4-element quaternion vector.

Parameters:

data 4-element quaternion vector.

Returns:

0 on success, an error code otherwise.

5.5.2.25 int_fast16_t inv_get_sample_frequency (void)

Returns the step size for quaternion type data.

Typically the data rate for each FIFO packet. When the gryos are sleeping this value will return the last value set by SetSampleStepSizeMs()

Returns:

step size for quaternion type data

5.5.2.26 int_fast16_t inv_get_sample_step_size_ms (void)

Returns the step size for quaternion type data.

Typically the data rate for each FIFO packet. When the gryos are sleeping this value will return the last value set by SetSampleStepSizeMs()

Returns:

step size for quaternion type data

5.5.2.27 inv_error_t inv_get_temperature (long * *data*)

Returns 1-element vector of temperature.

It is read from the hardware if it doesn't exist in the FIFO.

Parameters:

data 1-element vector of temperature

Returns:

0 on success or an error code.

5.5.2.28 inv_error_t inv_get_unquantized_accel (long * *data*)

Get the Decoded Accel Data.

Parameters:

data a buffer to store the quantized data.

Returns:

INV_SUCCESS if successful, a non-zero error code otherwise.

5.5.2.29 inv_error_t inv_init_fifo_param (void)

Initializes all the internal static variables for the FIFO module.

Note:

Should be called by the initialization routine such as inv_dmp_open().

Returns:

INV_SUCCESS if successful, a non-zero error code otherwise.

5.5.2.30 inv_error_t inv_read_and_process_fifo (int_fast8_t *numPackets*, int_fast8_t * *processed*)

Reads and processes FIFO data.

Also handles callbacks when data is ready.

Parameters:

numPackets Number of FIFO packets to try to read. You should use a large number here, such as 100, if you want to read all the full packets in the FIFO, which is typical operation.

processed The number of FIFO packets processed. This may be incremented even if high rate processes later fail.

Returns:

INV_SUCCESS if successful, a non-zero error code otherwise.

5.5.2.31 inv_error_t inv_send_accel (uint_fast16_t *elements*, uint_fast16_t *accuracy*)

Sends accelerometer data to the FIFO.

Parameters:

elements Which of the 3 elements to send. Use INV_ALL for 3 axis or INV_ELEMENT_1, INV_ELEMENT_2, INV_ELEMENT_3 or'd together for a subset.

accuracy Set to INV_32_BIT for 32-bit data, or INV_16_BIT for 16 bit data. Set to zero to remove it from the FIFO.

5.5.2.32 inv_error_t inv_send_cntrl_data (uint_fast16_t *elements*, uint_fast16_t *accuracy*)

Sends control data to the FIFO.

Control data is a 4 length vector of 32-bits.

Parameters:

elements Which of the 4 elements to send. Use INV_ALL for all or INV_ELEMENT_1, INV_ELEMENT_2, INV_ELEMENT_3, INV_ELEMENT_4 or'd together for a subset.

accuracy Set to INV_32_BIT for 32-bit data, or INV_16_BIT for 16 bit data. Set to zero to remove it from the FIFO.

5.5.2.33 inv_error_t inv_send_external_sensor_data (uint_fast16_t *elements*, uint_fast16_t *accuracy*)

Sends raw external data to the FIFO.

Should be called once after inv_dmp_open() and before inv_dmp_start().

Parameters:

elements Which of the 3 elements to send. Use INV_ALL for all of them or INV_ELEMENT_1, INV_ELEMENT_2, INV_ELEMENT_3 or'd together for a subset.

accuracy INV_16_BIT to send data, 0 to stop sending this data. Sending and Stop sending are reference counted, so data actually stops when the reference reaches zero.

5.5.2.34 inv_error_t inv_send_gravity (uint_fast16_t *elements*, uint_fast16_t *accuracy*)

Send the computed gravity vectors into the FIFO.

The gravity vectors can be retrieved from the FIFO via inv_get_gravity(), to have the gravitation vector expressed in coordinates relative to the body.

Gravity is a derived vector derived from the quaternion.

Parameters:

elements the gravitation vectors components bitmask. To send all compoents use INV_ALL.

accuracy The number of bits the gravitation vector is expressed into. Set to INV_-32_BIT for 32-bit data, or INV_16_BIT for 16 bit data. Set to zero to remove it from the FIFO.

Returns:

INV_SUCCESS if successful, a non-zero error code otherwise.

5.5.2.35 inv_error_t inv_send_gyro (uint_fast16_t *elements*, uint_fast16_t *accuracy*)

Sends gyro data to the FIFO.

Gyro data is a 3 length vector of 32-bits. Should be called once after inv_dmp_open() and before inv_dmp_start().

Parameters:

elements Which of the 3 elements to send. Use INV_ALL for all of them or INV_ELEMENT_1, INV_ELEMENT_2, INV_ELEMENT_3 or'd together for a subset.

accuracy Set to INV_32_BIT for 32-bit data, or INV_16_BIT for 16 bit data. Set to zero to remove it from the FIFO.

5.5.2.36 inv_error_t inv_send_linear_accel (uint_fast16_t *elements*, uint_fast16_t *accuracy*)

Sends linear accelerometer data to the FIFO.

Linear accelerometer data is a 3 length vector of 32-bits. It is the acceleration in the body frame with gravity removed.

Parameters:

elements Which of the 3 elements to send. Use INV_ALL for all of them or INV_ELEMENT_1, INV_ELEMENT_2, INV_ELEMENT_3 or'd together for a subset.

NOTE: Elements is ignored if the fifo rate is < INV_MAX_NUM_ACCEL_-SAMPLES

Parameters:

accuracy Set to INV_32_BIT for 32-bit data, or INV_16_BIT for 16 bit data. Set to zero to remove it from the FIFO.

5.5.2.37 inv_error_t inv_send_linear_accel_in_world (uint_fast16_t *elements*, uint_fast16_t *accuracy*)

Sends linear world accelerometer data to the FIFO.

Linear world accelerometer data is a 3 length vector of 32-bits. It is the acceleration in the world frame with gravity removed. Should be called once after inv_dmp_open() and before inv_dmp_start().

Parameters:

elements Which of the 3 elements to send. Use INV_ALL for all of them or INV_ELEMENT_1, INV_ELEMENT_2, INV_ELEMENT_3 or'd together for a subset.

accuracy Set to INV_32_BIT for 32-bit data, or INV_16_BIT for 16 bit data.

5.5.2.38 inv_error_t inv_send_packet_number (uint_fast16_t *accuracy*)

Adds a rolling counter to the fifo packet.

When used with the footer the data comes out the first time:

```
<data0><data1>...<dataN><PacketNum0><PacketNum1>
```

for every other packet it is

```
<FifoFooter0><FifoFooter1><data0><data1>...<dataN><PacketNum0><PacketNum1>
```

This allows for scanning of the fifo for packets

Returns:

INV_SUCCESS or error code

5.5.2.39 inv_error_t inv_send_quantized_accel (uint_fast16_t *elements*, uint_fast16_t *accuracy*)

Send the Quantized Acceleromter data into the FIFO.

The data can be retrieved using inv_get_quantized_accel() or inv_get_unquantized_accel().

To be useful this should be set to fifo_rate + 1 if less than INV_MAX_NUM_ACCEL_SAMPLES, otherwise it doesn't work.

Parameters:

elements the components bitmask. To send all compoents use INV_ALL.

accuracy Use INV_32_BIT for 32-bit data or INV_16_BIT for 16-bit data. Set to zero to remove it from the FIFO.

Returns:

INV_SUCCESS if successful, a non-zero error code otherwise.

5.5.2.40 inv_error_t inv_send_quaternion (uint_fast16_t *accuracy*)

Sends quaternion data to the FIFO.

Quaternion data is a 4 length vector of 32-bits. Should be called once after inv_dmp_open() and before inv_dmp_start().

Parameters:

accuracy Set to INV_32_BIT for 32-bit data, or INV_16_BIT for 16 bit data.

5.5.2.41 inv_error_t inv_send_sensor_data (uint_fast16_t *elements*, uint_fast16_t *accuracy*)

Sends raw data to the FIFO.

Should be called once after inv_dmp_open() and before inv_dmp_start().

Parameters:

elements Which of the 7 elements to send. Use INV_ALL for all of them or INV_ELEMENT_1, INV_ELEMENT_2, INV_ELEMENT_3 ... INV_ELEMENT_7 or'd together for a subset. The first element is temperature, the next 3 are gyro data, and the last 3 accel data.

accuracy The element's accuracy, can be INV_16_BIT, INV_32_BIT, or 0 to turn off.

Returns:

0 if successful, a non-zero error code otherwise.

5.5.2.42 inv_error_t inv_set_fifo_processed_callback (void(∗)(void) *func*)

inv_set_fifo_processed_callback is used to set a processed data callback function.

inv_set_fifo_processed_callback sets a user defined callback function that triggers when all the decoding has been finished by the motion processing engines. It is called before other bigger processing engines to allow lower latency for the user.

Precondition:

inv_dmp_open()

or inv_open_low_power_pedometer() or inv_eis_open_dmp()

and inv_dmp_start() must **NOT** have been called.

Parameters:

func A user defined callback function.

Returns:

INV_SUCCESS if successful, or non-zero error code otherwise.

5.5.2.43 inv_error_t inv_set_fifo_rate (unsigned short *fifoRate*)

Command the MPU to put data in the FIFO at a particular rate.

The DMP will add fifo entries every fifoRate + 1 MPU cycles. For example if the MPU is running at 200Hz the following values apply:

fifoRate	DMP Sample Rate	FIFO update frequency
0	200Hz	200Hz
1	200Hz	100Hz
2	200Hz	50Hz
4	200Hz	40Hz
9	200Hz	20Hz
19	200Hz	10Hz

Note: if the DMP is running, (state == INV_STATE_DMP_STARTED) then inv_run_state_callbacks() will be called to allow features that depend upon fundamental constants to be updated.

Precondition:

inv_dmp_open()

or inv_open_low_power_pedometer() or inv_eis_open_dmp()

and inv_dmp_start() must **NOT** have been called.

Parameters:

fifoRate Divider value - 1. Output rate is (DMP Sample Rate) / (fifoRate + 1).

Returns:

INV_SUCCESS if successful, ML error code on any failure.

5.5.2.44 inv_error_t inv_set_gyro_data_source (uint_fast8_t *source*)

Set the gyro source to output to the fifo.

Parameters:

source The source. One of

- INV_GYRO_FROM_RAW
- INV_GYRO_FROM_QUATERNION

Returns:

INV_SUCCESS or non-zero error code;

5.5.2.45 inv_error_t inv_set_linear_accel_filter_coef (float *coef*)

Sets the filter coefficent used for computing the acceleration bias which is used to compute linear acceleration.

Parameters:

coef Fitler coefficient. 0. means no filter, a small number means a small cut-off frequency with an increasing number meaning an increasing cutoff frequency.

5.6 MLFIFO_HW

Motion Library - FIFO HW Driver.

Files

- file mlFIFOHW.c
  The Motion Library Fifo Hardware Layer.

Functions

- short inv_get_fifo_count (void)
  inv_get_fifo_count is used to get the number of bytes left in the FIFO.
- inv_error_t inv_get_fifo_status (void)
  Used to query the status of the FIFO.
- void inv_init_fifo_hardare (void)
  Initializes the internal FIFO data structure.
- inv_error_t inv_reset_fifo (void)
  Clears the FIFO status and its content.

5.6.1 Detailed Description

Motion Library - FIFO HW Driver.

Provides facilities to interact with the FIFO.

5.6.2 Function Documentation

5.6.2.1 short inv_get_fifo_count (void)

inv_get_fifo_count is used to get the number of bytes left in the FIFO.

This function returns the stored value and does not access the hardware. See inv_get_-fifo_length().

Returns:

the number of bytes left in the FIFO

5.6.2.2 inv_error_t inv_get_fifo_status (void)

Used to query the status of the FIFO.

Returns:

INV_SUCCESS if the fifo is OK. An error code otherwise.

5.6.2.3 inv_error_t inv_reset_fifo (void)

Clears the FIFO status and its content.

Note:

Halt the DMP writing into the FIFO for the time needed to reset the FIFO.

Returns:

INV_SUCCESS if successful, a non-zero error code otherwise.

5.7 ML_SUPERVISOR

Basic sensor fusion supervisor functionalities.

Files

- file mlsupervisor.c

  Basic sensor fusion supervisor functionalities.

Functions

- void inv_init_sensor_fusion_supervisor (void)

  This initializes all variables that should be reset on.

- inv_error_t inv_pressure_supervisor (void)

  Entry point for software sensor fusion operations.

5.7.1 Detailed Description

Basic sensor fusion supervisor functionalities.

5.7.2 Function Documentation

5.7.2.1 inv_error_t inv_pressure_supervisor (void)

Entry point for software sensor fusion operations.

Manages hardware interaction, calls sensor fusion supervisor for bias calculation.

Returns:

INV_SUCCESS or non-zero error code on error.

5.8 MLDL

Motion Library - Driver Layer.

Files

- file mldl.c

  The Motion Library Driver Layer.

- file mldl_cfg.c

  The Motion Library Driver Layer.

- file mldl_cfg_mpu.c

  The Motion Library Driver Layer.

Functions

- void inv_clear_interrupt_trigger (unsigned char srcIndex)

  clear the 'triggered' status for an interrupt source.

- inv_error_t inv_clock_source (unsigned char clkSource)

  inv_clock_source function sets the clock source for the MPU gyro processing.

- inv_error_t inv_dl_cfg_sampling (unsigned char lpf, unsigned char divider)

  configures the output sampling rate on the MPU.

- inv_error_t inv_dl_close (void)

  Closes/Cleans up the ML Driver Layer.

- inv_error_t inv_dl_open (void *mlsl_handle)

  Open the driver layer and resets the internal gyroscope, accelerometer, and compass data structures.

- inv_error_t inv_dl_start (unsigned long sensors)

  Starts the DMP running.

- inv_error_t inv_dl_stop (unsigned long sensors)

  Stops the DMP running and puts it in low power as requested.

- struct mldl_cfg * inv_get_dl_config (void)

Get a pointer to the internal data structure storing the configuration for the MPU, the accelerometer and the compass in use.

- inv_error_t inv_get_interrupt_status (unsigned char intPin, unsigned char *status)

  inv_get_interrupt_status returns the interrupt status from the specified interrupt pin.

- unsigned char inv_get_interrupt_trigger (unsigned char srcIndex)

  query the current status of an interrupt source.

- unsigned char inv_get_mpu_slave_addr (void)

  Query the MPU slave address.

- unsigned char inv_get_product_rev (void)

  Get the product revision ID.

- unsigned char inv_get_silicon_rev (void)

  Get the silicon revision ID.

- inv_error_t inv_init_requested_sensors (unsigned long sensors)

  Sets the requested_sensors.

- inv_error_t inv_interrupt_handler (unsigned char intSource)

  inv_interrupt_handler function should be called when an interrupt is received.

- inv_error_t inv_load_dmp (const unsigned char *buffer, unsigned short length, unsigned short config)

  Load the DMP with the given code and configuration.

- int inv_mpu_close (struct mldl_cfg *mldl_cfg, void *gyro_handle, void *accel_-handle, void *compass_handle, void *pressure_handle)

  Close the mpu interface.

- int inv_mpu_get_slave_config (struct mldl_cfg *mldl_cfg, void *gyro_handle, void *slave_handle, struct ext_slave_config *data, struct ext_slave_descr *slave, struct ext_slave_platform_data *pdata)

  Request slave configuration information.

- int inv_mpu_open (struct mldl_cfg *mldl_cfg, void *gyro_handle, void *accel_-handle, void *compass_handle, void *pressure_handle)

  Initializes the pdata structure to defaults.

- int inv_mpu_resume (struct mldl_cfg *mldl_cfg, void *gyro_handle, void *accel_handle, void *compass_handle, void *pressure_handle, unsigned long sensors)

  resume the MPU device and all the other sensor devices from their low power state.

- int inv_mpu_set_firmware (struct mldl_cfg *mldl_cfg, void *mlsl_handle, const unsigned char *data, int size)

  Sets the firmware cache.

- int inv_mpu_slave_config (struct mldl_cfg *mldl_cfg, void *gyro_handle, void *slave_handle, struct ext_slave_config *data, struct ext_slave_descr *slave, struct ext_slave_platform_data *pdata)

  Send slave configuration information.

- int inv_mpu_slave_read (struct mldl_cfg *mldl_cfg, void *gyro_handle, void *slave_handle, struct ext_slave_descr *slave, struct ext_slave_platform_data *pdata, unsigned char *data)

  Send slave configuration information.

- int inv_mpu_suspend (struct mldl_cfg *mldl_cfg, void *gyro_handle, void *accel_handle, void *compass_handle, void *pressure_handle, unsigned long sensors)

  suspend the MPU device and all the other sensor devices into their low power state.

- inv_error_t inv_set_dl_cfg_int (unsigned char triggers)

  inv_set_dl_cfg_int configures the interrupt function on the specified pin.

- inv_error_t inv_set_external_sync (unsigned char extSync)

  This function sets the external sync for the MPU sampling.

- inv_error_t inv_set_full_scale (float fullScale)

  set the full scale range for the gyros.

- inv_error_t inv_set_offset (const short *offset)

  Set the gyro offset.

- inv_error_t inv_set_offsetTC (const unsigned char *tc)

  Set the Temperature Compensation offset.

5.8.1 Detailed Description

Motion Library - Driver Layer.

The Motion Library Driver Layer provides the intrface to the system drivers that are used by the Motion Library.

5.8.2 Function Documentation

5.8.2.1 void inv_clear_interrupt_trigger (unsigned char *srcIndex*)

clear the 'triggered' status for an interrupt source.

Parameters:

srcIndex index of the interrupt source. Currently only INTPIN_MPU is supported.

5.8.2.2 inv_error_t inv_clock_source (unsigned char *clkSource*)

inv_clock_source function sets the clock source for the MPU gyro processing.

The source can be any of the following:

- Internal 8MHz oscillator,
- PLL with X gyro as reference,
- PLL with Y gyro as reference,
- PLL with Z gyro as reference,
- PLL with external 32.768Mhz reference, or
- PLL with external 19.2MHz reference

For best accuracy and timing, it is highly recommended to use one of the gyros as the clock source; however this gyro must be enabled to use its clock (see 'MLDLPowerMgmtMPU()').

Parameters:

clkSource Clock source selection. Can be one of:

- CLK_INTERNAL,
- CLK_PLLGYROX,
- CLK_PLLGYROY,
- CLK_PLLGYROZ,
- CLK_PLLEXT32K, or
- CLK_PLLEXT19M.

Returns:

Zero if the command is successful; an error code otherwise.

5.8.2.3 inv_error_t inv_dl_cfg_sampling (unsigned char *lpf*, unsigned char *divider*)

configures the output sampling rate on the MPU.

Three parameters control the sampling:

1) Low pass filter bandwidth, and 2) output sampling divider.

The output sampling rate is determined by the divider and the low pass filter setting. If the low pass filter is set to 'MPUFILTER_256HZ_NOLPF2', then the sample rate going into the divider is 8kHz; for all other settings it is 1kHz. The 8-bit divider will divide this frequency to get the resulting sample frequency. For example, if the filter setting is not 256Hz and the divider is set to 7, then the sample rate is as follows: sample rate = internal sample rate / div = 1kHz / 8 = 125Hz (or 8ms).

The low pass filter selection codes control both the cutoff frequency of the internal low pass filter and internal analog sampling rate. The latter, in turn, affects the final output sampling rate according to the sample rate divider settig. 0 -> 256 Hz cutoff BW, 8 kHz analog sample rate, 1 -> 188 Hz cutoff BW, 1 kHz analog sample rate, 2 -> 98 Hz cutoff BW, 1 kHz analog sample rate, 3 -> 42 Hz cutoff BW, 1 kHz analog sample rate, 4 -> 20 Hz cutoff BW, 1 kHz analog sample rate, 5 -> 10 Hz cutoff BW, 1 kHz analog sample rate, 6 -> 5 Hz cutoff BW, 1 kHz analog sample rate, 7 -> 2.1 kHz cutoff BW, 8 kHz analog sample rate.

Parameters:

lpf low pass filter, 0 to 7.

divider Output sampling rate divider, 0 to 255.

Returns:

ML_SUCESS if successful; a non-zero error code otherwise.

5.8.2.4 inv_error_t inv_dl_close (void)

Closes/Cleans up the ML Driver Layer.

Put the device in sleep mode.

Returns:

INV_SUCCESS or non-zero error code.

5.8.2.5 inv_error_t inv_dl_open (void * *mlsl_handle*)

Open the driver layer and resets the internal gyroscope, accelerometer, and compass data structures.

Parameters:

mlslHandle the serial handle.

Returns:

INV_SUCCESS if successful, a non-zero error code otherwise.

5.8.2.6 inv_error_t inv_dl_start (unsigned long *sensors*)

Starts the DMP running.

Resumes the sensor if any of the sensor axis or components are requested

Parameters:

sensors Bitfield of the sensors to turn on. Combination of the following:

- INV_X_GYRO
- INV_Y_GYRO
- INV_Z_GYRO
- INV_DMP_PROCESSOR
- INV_X_ACCEL
- INV_Y_ACCEL
- INV_Z_ACCEL
- INV_X_COMPASS
- INV_Y_COMPASS
- INV_Z_COMPASS
- INV_X_PRESSURE
- INV_Y_PRESSURE
- INV_Z_PRESSURE
- INV_THREE_AXIS_GYRO
- INV_THREE_AXIS_ACCEL
- INV_THREE_AXIS_COMPASS
- INV_THREE_AXIS_PRESSURE

Returns:

INV_SUCCESS or non-zero error code

5.8.2.7 inv_error_t inv_dl_stop (unsigned long *sensors*)

Stops the DMP running and puts it in low power as requested.

Suspends each sensor according to the bitfield, if all axis and components of the sensor is off.

Parameters:

sensors Bitfiled of the sensors to leave on. Combination of the following:

- INV_X_GYRO
- INV_Y_GYRO
- INV_Z_GYRO
- INV_X_ACCEL
- INV_Y_ACCEL
- INV_Z_ACCEL
- INV_X_COMPASS
- INV_Y_COMPASS
- INV_Z_COMPASS
- INV_X_PRESSURE
- INV_Y_PRESSURE
- INV_Z_PRESSURE
- INV_THREE_AXIS_GYRO
- INV_THREE_AXIS_ACCEL
- INV_THREE_AXIS_COMPASS
- INV_THREE_AXIS_PRESSURE

Returns:

INV_SUCCESS or non-zero error code

5.8.2.8 struct mldl_cfg* inv_get_dl_config (void) [read]

Get a pointer to the internal data structure storing the configuration for the MPU, the accelerometer and the compass in use.

Returns:

a pointer to the data structure of type 'struct mldl_cfg'.

5.8.2.9 inv_error_t inv_get_interrupt_status (unsigned char *intPin*, unsigned char * *status*)

inv_get_interrupt_status returns the interrupt status from the specified interrupt pin.

Parameters:

intPin Currently only the value INTPIN_MPU is supported.

status The available statuses are:

- BIT_MPU_RDY_EN
- BIT_DMP_INT_EN
- BIT_RAW_RDY_EN

Returns:

INV_SUCCESS or a non-zero error code.

5.8.2.10 unsigned char inv_get_interrupt_trigger (unsigned char *srcIndex*)

query the current status of an interrupt source.

Parameters:

srcIndex index of the interrupt source. Currently the only source supported is INTPIN_MPU.

Returns:

1 if the interrupt has been triggered.

5.8.2.11 unsigned char inv_get_mpu_slave_addr (void)

Query the MPU slave address.

Returns:

The 7-bit mpu slave address.

5.8.2.12 unsigned char inv_get_product_rev (void)

Get the product revision ID.

Returns:

The product revision ID (0 will be read if inv_mpu_open returned an error)

5.8.2.13 unsigned char inv_get_silicon_rev (void)

Get the silicon revision ID.

Returns:

The silicon revision ID (0 will be read if inv_mpu_open returned an error)

5.8.2.14 inv_error_t inv_init_requested_sensors (unsigned long *sensors*)

Sets the requested_sensors.

Accessor to set the requested_sensors field of the mldl_cfg structure. Typically set at initialization.

Parameters:

sensors Bitfield of the sensors that are going to be used. Combination of the following:

- INV_X_GYRO
- INV_Y_GYRO
- INV_Z_GYRO
- INV_DMP_PROCESSOR
- INV_X_ACCEL
- INV_Y_ACCEL
- INV_Z_ACCEL
- INV_X_COMPASS
- INV_Y_COMPASS
- INV_Z_COMPASS
- INV_X_PRESSURE
- INV_Y_PRESSURE
- INV_Z_PRESSURE
- INV_THREE_AXIS_GYRO
- INV_THREE_AXIS_ACCEL
- INV_THREE_AXIS_COMPASS
- INV_THREE_AXIS_PRESSURE

Returns:

INV_SUCCESS or non-zero error code

5.8.2.15 inv_error_t inv_interrupt_handler (unsigned char *intSource*)

inv_interrupt_handler function should be called when an interrupt is received.

The source parameter identifies which interrupt source caused the interrupt. Note that this routine should not be called directly from the interrupt service routine.

Parameters:

intSource MPU, AUX1, AUX2, or timer. Can be one of: INTSRC_MPU, INTSRC_AUX1, INTSRC_AUX2, or INT_SRC_TIMER.

Returns:

Zero if the command is successful; an error code otherwise.

5.8.2.16 inv_error_t inv_load_dmp (const unsigned char * *buffer*, unsigned short *length*, unsigned short *config*)

Load the DMP with the given code and configuration.

Parameters:

buffer the DMP data.

length the length in bytes of the DMP data.

config the DMP configuration.

Returns:

INV_SUCCESS if successful, a non-zero error code otherwise.

5.8.2.17 int inv_mpu_close (struct mldl_cfg * *mldl_cfg*, void * *mlsl_handle*, void * *accel_handle*, void * *compass_handle*, void * *pressure_handle*)

Close the mpu interface.

Stub for driver close.

pointer to the configuration structure pointer to the serial layer handle

Returns:

INV_SUCCESS or non-zero error code

Just verify that the devices are suspended

Parameters:

mldl_cfg handle to the config structure

mlsl_handle handle to the mpu serial layer

accel_handle handle to the accel serial layer

compass_handle handle to the compass serial layer

pressure_handle handle to the compass serial layer

Returns:

INV_SUCCESS or non-zero error code

5.8.2.18 int inv_mpu_get_slave_config (struct mldl_cfg * *mldl_cfg*, void * *gyro_handle*, void * *slave_handle*, struct ext_slave_config * *data*, struct ext_slave_descr * *slave*, struct ext_slave_platform_data * *pdata*)

Request slave configuration information.

Use this specifically after requesting a slave configuration to see what the slave accually accepted.

Parameters:

mldl_cfg pointer to the mldl configuration structure

gyro_handle handle to the gyro sensor

slave_handle handle to the slave sensor

data the data being requested.

slave slave description

pdata slave platform data

Returns:

0 or non-zero error code

5.8.2.19 int inv_mpu_open (struct mldl_cfg * *mldl_cfg*, void * *mlsl_handle*, void * *accel_handle*, void * *compass_handle*, void * *pressure_handle*)

Initializes the pdata structure to defaults.

Opens the device to read silicon revision, product id and whoami.

The internal device configuration data structure. The serial communication handle.

Returns:

INV_SUCCESS if silicon revision, product id and woami are supported by this software.

Opens the device to read silicon revision, product id and whoami. Leaves the device in suspended state for low power.

Parameters:

mldl_cfg handle to the config structure

mlsl_handle handle to the mpu serial layer

accel_handle handle to the accel serial layer

compass_handle handle to the compass serial layer

pressure_handle handle to the pressure serial layer

Returns:

INV_SUCCESS if silicon revision, product id and woami are supported by this software.

5.8.2.20 int inv_mpu_resume (struct mldl_cfg * *mldl_cfg*, void * *mlsl_handle*, void * *accel_handle*, void * *compass_handle*, void * *pressure_handle*, unsigned long *sensors*)

resume the MPU device and all the other sensor devices from their low power state.

pointer to the configuration structure the main file handle to the MPU device. an handle to the accelerometer device, if sitting onto a separate bus. Can match mlsl_handle if the accelerometer device operates on the same primary bus of MPU. an handle to the compass device, if sitting onto a separate bus. Can match mlsl_handle if the compass device operates on the same primary bus of MPU. an handle to the pressure sensor device, if sitting onto a separate bus. Can match mlsl_handle if the pressure sensor device operates on the same primary bus of MPU. whether resuming the gyroscope device is actually needed (if the device supports low power mode of some sort). whether resuming the accelerometer device is actually needed (if the device supports low power mode of some sort). whether resuming the compass device is actually needed (if the device supports low power mode of some sort). whether resuming the pressure sensor device is actually needed (if the device supports low power mode of some sort).

Returns:

INV_SUCCESS or a non-zero error code.

pointer to the configuration structure the main file handle to the MPU device. an handle to the accelerometer device, if sitting onto a separate bus. Can match mlsl_handle if

the accelerometer device operates on the same primary bus of MPU. an handle to the compass device, if sitting onto a separate bus. Can match mlsl_handle if the compass device operates on the same primary bus of MPU. an handle to the pressure sensor device, if sitting onto a separate bus. Can match mlsl_handle if the pressure sensor device operates on the same primary bus of MPU. sensor enable mask requested.

Returns:

INV_SUCCESS or a non-zero error code.

5.8.2.21 int inv_mpu_set_firmware (struct mldl_cfg * *mldl_cfg*, void * *mlsl_handle*, const unsigned char * *data*, int *size*)

Sets the firmware cache.

Parameters:

mldl_cfg pointer to the configuration

mlsl_handle serial handle

data firmware

size sizeof the firmware

Returns:

INV_SUCCESS or non-zero error code

5.8.2.22 int inv_mpu_slave_config (struct mldl_cfg * *mldl_cfg*, void * *gyro_handle*, void * *slave_handle*, struct ext_slave_config * *data*, struct ext_slave_descr * *slave*, struct ext_slave_platform_data * *pdata*)

Send slave configuration information.

Parameters:

mldl_cfg pointer to the mldl configuration structure

gyro_handle handle to the gyro sensor

slave_handle handle to the slave sensor

data the data being sent

slave slave description

pdata slave platform data

Returns:

0 or non-zero error code

5.8.2.23 int inv_mpu_slave_read (struct mldl_cfg * *mldl_cfg*, void * *gyro_handle*, void * *slave_handle*, struct ext_slave_descr * *slave*, struct ext_slave_platform_data * *pdata*, unsigned char * *data*)

Send slave configuration information.

Parameters:

mldl_cfg pointer to the mldl configuration structure

gyro_handle handle to the gyro sensor

slave_handle handle to the slave sensor

slave slave description

pdata slave platform data

data where to store the read data

Returns:

0 or non-zero error code

5.8.2.24 int inv_mpu_suspend (struct mldl_cfg * *mldl_cfg*, void * *gyro_handle*, void * *accel_handle*, void * *compass_handle*, void * *pressure_handle*, unsigned long *sensors*)

suspend the MPU device and all the other sensor devices into their low power state.

a pointer to the struct mldl_cfg internal data structure. the main file handle to the MPU device. an handle to the accelerometer device, if sitting onto a separate bus. Can match gyro_handle if the accelerometer device operates on the same primary bus of MPU. an handle to the compass device, if sitting onto a separate bus. Can match gyro_handle if the compass device operates on the same primary bus of MPU. an handle to the pressure sensor device, if sitting onto a separate bus. Can match gyro_handle if the pressure sensor device operates on the same primary bus of MPU. whether suspending the accelerometer device is actually needed (if the device supports low power mode of some sort). whether suspending the compass device is actually needed (if the device supports low power mode of some sort). whether suspending the pressure sensor device is actually needed (if the device supports low power mode of some sort).

Returns:

INV_SUCCESS or a non-zero error code.

5.8.2.25 inv_error_t inv_set_dl_cfg_int (unsigned char *triggers*)

inv_set_dl_cfg_int configures the interrupt function on the specified pin.

The basic interrupt signal characteristics can be set (i.e. active high/low, open drain/-push pull, etc.) and the triggers can be set. Currently only INTPIN_MPU is supported.

Parameters:

triggers bitmask of triggers to enable for interrupt. The available triggers are:

- BIT_MPU_RDY_EN
- BIT_DMP_INT_EN
- BIT_RAW_RDY_EN

Returns:

Zero if the command is successful, an error code otherwise.

5.8.2.26 inv_error_t inv_set_external_sync (unsigned char *extSync*)

This function sets the external sync for the MPU sampling.

It can be synchronized on the LSB of any of the gyros, any of the external accels, or on the temp readings.

Parameters:

extSync External sync selection, 0 to 7.

Returns:

Zero if the command is successful; an error code otherwise.

5.8.2.27 inv_error_t inv_set_full_scale (float *fullScale*)

set the full scale range for the gyros.

The full scale selection codes correspond to: 0 -> 250 dps, 1 -> 500 dps, 2 -> 1000 dps, 3 -> 2000 dps. Full scale range affect the MPU's measurement sensitivity.

Parameters:

fullScale the gyro full scale range in dps.

Returns:

INV_SUCCESS or non-zero error code.

5.8.2.28 inv_error_t inv_set_offset (const short * *offset*)

Set the gyro offset.

Parameters:

offset a pointer to the gyro offset for the 3 gyro axes. This is scaled as it would be written to the hardware registers.

Returns:

INV_SUCCESS if successful, a non-zero error code otherwise.

5.8.2.29 inv_error_t inv_set_offsetTC (const unsigned char * *tc*)

Set the Temperature Compensation offset.

Parameters:

tc a pointer to the temperature compensations offset for the 3 gyro axes.

Returns:

INV_SUCCESS if successful, a non-zero error code otherwise.

5.9 CONTROL

Motion Library - Control Engine.

Files

- file mlcontrol.c

  The Control Library.

Typedefs

- typedef void(∗ fpGridCb)(unsigned short controlSignal, long ∗gridNum, long ∗gridChange)

  GridCallback function pointer type, to be passed as argument of inv_set_grid_callback.

Functions

- inv_error_t inv_disable_control (void)

  Disables the INV_CONTROL engine.

- inv_error_t inv_enable_control (void)

  Enables the INV_CONTROL engine.

- inv_error_t inv_get_control_data (long ∗controlSignal, long ∗gridNum, long ∗gridChange)

  inv_get_control_data is used to get the current control data.

- inv_error_t inv_get_control_signal (unsigned short controlSignal, unsigned short reset, long ∗data)

  inv_get_control_signal is used to get the current control signal with high precision.

- inv_error_t inv_get_grid_num (unsigned short controlSignal, unsigned short reset, long ∗data)

  inv_get_grid_num is used to get the current grid location for a certain control signal.

- inv_error_t inv_set_control_data (unsigned short controlSignal, unsigned short parameterArray, unsigned short parameterAxis)

  inv_set_control_data is used to assign physical parameters to control signals.

- inv_error_t inv_set_control_func (unsigned short function)

  inv_set_control_func allows the user to choose how the sensor data will be processed in order to provide a control parameter.

- inv_error_t inv_set_control_sensitivity (unsigned short controlSignal, long sensitivity)

  inv_set_control_sensitivity is used to set the sensitivity for a control signal.

- inv_error_t inv_set_grid_callback (fpGridCb func)

  inv_set_grid_callback is used to register a callback function that will trigger when the grid location changes.

- inv_error_t inv_set_grid_max (unsigned short controlSignal, long maximum)

  inv_set_grid_max is used to set the maximum grid number for a control signal.

- inv_error_t inv_set_grid_thresh (unsigned short controlSignal, long threshold)

  inv_set_grid_thresh is used to set the grid size for a control signal.

5.9.1 Detailed Description

Motion Library - Control Engine.

The Control Library processes gyroscopes, accelerometers, and compasses to provide control signals that can be used in user interfaces. These signals can be used to manipulate objects such as documents, images, cursors, menus, etc.

5.9.2 Typedef Documentation

5.9.2.1 typedef void(∗ fpGridCb)(unsigned short controlSignal, long ∗gridNum, long ∗gridChange)

GridCallback function pointer type, to be passed as argument of inv_set_grid_callback.

Parameters:

controlSignal Indicates which control signal crossed a grid threshold. Must be one of:

- INV_CONTROL_1,
- INV_CONTROL_2,
- INV_CONTROL_3 and
- INV_CONTROL_4.

gridNumber An array of four numbers representing the grid number for each control signal.

gridChange An array of four numbers representing the change in grid number for each control signal.

5.9.3 Function Documentation

5.9.3.1 inv_error_t inv_disable_control (void)

Disables the INV_CONTROL engine.

Note:

This function replaces MLDisable(INV_CONTROL)

Precondition:

inv_dmp_open() with MLDmpDefaultOpen or MLDmpPedometerStandAlone() must have been called.

Returns:

INV_SUCCESS or non-zero error code

5.9.3.2 inv_error_t inv_enable_control (void)

Enables the INV_CONTROL engine.

Note:

This function replaces MLEnable(INV_CONTROL)

Precondition:

inv_dmp_open() with MLDmpDefaultOpen or MLDmpPedometerStandAlone() must have been called.

Returns:

INV_SUCCESS or non-zero error code

5.9.3.3 inv_error_t inv_get_control_data (long * *controlSignal*, long * *gridNum*, long * *gridChange*)

inv_get_control_data is used to get the current control data.

Precondition:

inv_dmp_open() Must be called with MLDmpDefaultOpen() or inv_open_low_-power_pedometer().

Parameters:

controlSignal Indicates which control signal is being queried. Must be one of:

- INV_CONTROL_1,
- INV_CONTROL_2,
- INV_CONTROL_3 or
- INV_CONTROL_4.

gridNum A pointer to pass gridNum info back to the user.

gridChange A pointer to pass gridChange info back to the user.

Returns:

Zero if the command is successful; an ML error code otherwise.

5.9.3.4 inv_error_t inv_get_control_signal (unsigned short *controlSignal*, unsigned short *reset*, long * *data*)

inv_get_control_signal is used to get the current control signal with high precision.

inv_get_control_signal is used to acquire the current data of a control signal. If INV_-GRID is being used, inv_get_grid_number will probably be preferrable.

Parameters:

controlSignal Indicates which control signal is being queried. Must be one of:

- INV_CONTROL_1,
- INV_CONTROL_2,
- INV_CONTROL_3 or
- INV_CONTROL_4.

reset Indicates whether the control signal should be reset to zero. Options are INV_RESET or INV_NO_RESET

data A pointer to the current control signal data.

Returns:

Zero if the command is successful; an ML error code otherwise.

5.9.3.5 inv_error_t inv_get_grid_num (unsigned short *controlSignal*, unsigned short *reset*, long * *data*)

inv_get_grid_num is used to get the current grid location for a certain control signal.

inv_get_grid_num is used to acquire the current grid location.

Precondition:

inv_dmp_open() Must be called with MLDmpDefaultOpen() or inv_open_low_-power_pedometer().

Parameters:

controlSignal Indicates which control signal is being queried. Must be one of:

- INV_CONTROL_1,
- INV_CONTROL_2,
- INV_CONTROL_3 or
- INV_CONTROL_4.

reset Indicates whether the control signal should be reset to zero. Options are INV_RESET or INV_NO_RESET

data A pointer to the current grid number.

Returns:

Zero if the command is successful; an ML error code otherwise.

5.9.3.6 inv_error_t inv_set_control_data (unsigned short *controlSignal*, unsigned short *parameterArray*, unsigned short *parameterAxis*)

inv_set_control_data is used to assign physical parameters to control signals.

inv_set_control_data allows flexibility in assigning physical parameters to control signals. For example, the user is allowed to use raw gyroscope data as an input to the control algorithm. Alternatively, angular velocity can be used, which combines gyroscopes and accelerometers to provide a more robust physical parameter. Finally, angular velocity in world coordinates can be used, providing a control signal in which pitch and yaw are provided relative to gravity.

Precondition:

inv_dmp_open() Must be called with MLDmpDefaultOpen() or inv_open_low_-power_pedometer().

Parameters:

controlSignal Indicates which control signal is being modified. Must be one of:

- INV_CONTROL_1,
- INV_CONTROL_2,
- INV_CONTROL_3 or
- INV_CONTROL_4.

parameterArray Indicates which parameter array is being assigned to a control signal. Must be one of:

- INV_GYROS,
- INV_ANGULAR_VELOCITY, or

parameterAxis Indicates which axis of the parameter array will be used. Must be:

- INV_ROLL,
- INV_PITCH, or
- INV_YAW.

5.9.3.7 inv_error_t inv_set_control_func (unsigned short *function*)

inv_set_control_func allows the user to choose how the sensor data will be processed in order to provide a control parameter.

inv_set_control_func allows the user to choose which control functions will be incorporated in the sensor data processing. The control functions are:

- INV_GRID Indicates that the user will be controlling a system that has discrete steps, such as icons, menu entries, pixels, etc.
- INV_SMOOTH Indicates that noise from unintentional motion should be filtered out.
- INV_DEAD_ZONE Indicates that a dead zone should be used, below which sensor data is set to zero.
- INV_HYSTERESIS Indicates that, when INV_GRID is selected, hysteresis should be used to prevent the control signal from switching rapidly across elements of the grid.

Precondition:

inv_dmp_open() Must be called with MLDmpDefaultOpen() or inv_open_low_power_pedometer().

Parameters:

function Indicates what functions will be used. Can be a bitwise OR of several values.

Returns:

Zero if the command is successful; an ML error code otherwise.

5.9.3.8 inv_error_t inv_set_control_sensitivity (unsigned short *controlSignal*, long *sensitivity*)

inv_set_control_sensitivity is used to set the sensitivity for a control signal.

Precondition:

inv_dmp_open() Must be called with MLDmpDefaultOpen() or inv_open_low_-power_pedometer().

Parameters:

controlSignal Indicates which control signal is being modified. Must be one of:

- INV_CONTROL_1,
- INV_CONTROL_2,
- INV_CONTROL_3 or
- INV_CONTROL_4.

sensitivity The sensitivity of the control signal.

Returns:

error code

5.9.3.9 inv_error_t inv_set_grid_callback (fpGridCb *func*)

inv_set_grid_callback is used to register a callback function that will trigger when the grid location changes.

inv_set_grid_callback allows a user to define a callback function that will run when a control signal crosses a grid threshold.

Precondition:

inv_dmp_open() Must be called with MLDmpDefaultOpen() or inv_open_low_-power_pedometer(). inv_dmp_start must **NOT** have been called.

Parameters:

func A user defined callback function

Returns:

Zero if the command is successful; an ML error code otherwise.

5.9.3.10 inv_error_t inv_set_grid_max (unsigned short *controlSignal*, long *maximum*)

inv_set_grid_max is used to set the maximum grid number for a control signal.

inv_set_grid_max is used to adjust the maximum allowed grid number, above which the grid number will not be incremented. The minimum grid number is always zero.

Precondition:

inv_dmp_open() Must be called with MLDmpDefaultOpen() or inv_open_low_-power_pedometer().

Parameters:

controlSignal Indicates which control signal is being modified. Must be one of:

- INV_CONTROL_1,
- INV_CONTROL_2,
- INV_CONTROL_3 and
- INV_CONTROL_4.

maximum The maximum grid number for a control signal.

Returns:

Zero if the command is successful; an ML error code otherwise.

5.9.3.11 inv_error_t inv_set_grid_thresh (unsigned short *controlSignal*, long *threshold*)

inv_set_grid_thresh is used to set the grid size for a control signal.

inv_set_grid_thresh is used to adjust the size of the grid being controlled.

Parameters:

controlSignal Indicates which control signal is being modified. Must be one of:

- INV_CONTROL_1,
- INV_CONTROL_2,
- INV_CONTROL_3 and
- INV_CONTROL_4.

threshold The threshold of the control signal at which the grid number will be incremented or decremented.

Returns:

Zero if the command is successful; an ML error code otherwise.

5.10 ACCELDL

Motion Library - Accel Driver Layer.

Files

- file accel.c
  Accel setup and handling methods.
- file adxl34x.c
  Accelerometer setup and handling methods for AD adxl345 and adxl346.
- file bma150.c
  Accelerometer setup and handling methods for Bosch BMA150.
- file bma250.c
  Accelerometer setup and handling methods for Bosch BMA250.
- file kxsd9.c
  Accelerometer setup and handling methods for Kionix KXSD9.
- file kxtf9.c
  Accelerometer setup and handling methods for Kionix KXTF9.
- file lis331.c
  Accelerometer setup and handling methods for ST LIS331DLH.
- file lis3dh.c
  Accelerometer setup and handling methods for ST LIS3DH.
- file lsm303dlx_a.c
  Accelerometer setup and handling methods for ST LSM303DLH or LSM303DLM accel.
- file mma8450.c
  Accelerometer setup and handling methods for Freescale MMA8450.
- file mma845x.c
  Accelerometer setup and handling methods for Freescale MMA845X.
- file mpu6050.c
  Accelerometer setup and handling methods for Invensense MPU6050.

Functions

- unsigned char inv_accel_present (void)
  Used to determine if an accel is configured and used by the MPL.
- inv_error_t inv_get_accel_data (long *data)
  Get a sample of accel data from the device.
- unsigned short inv_get_accel_id (void)
  Get the ID of the accel in use.
- unsigned char inv_get_slave_addr (void)
  Query the accel slave address.

5.10.1 Detailed Description

Motion Library - Accel Driver Layer.

Accelerometer setup and handling methods for Kionix KXTF9.

Accelerometer setup and handling methods for Kionix KXSD9.

Provides the interface to setup and handle an accelerometer.

Provides the interface to setup and handle an accel connected to either the primary or the seconday I2C interface of the gyroscope.

5.10.2 Function Documentation

5.10.2.1 unsigned char inv_accel_present (void)

Used to determine if an accel is configured and used by the MPL.

Returns:

INV_SUCCESS if the accel is present.

5.10.2.2 inv_error_t inv_get_accel_data (long * *data*)

Get a sample of accel data from the device.

Parameters:

data the buffer to store the accel raw data for X, Y, and Z axes.

Returns:

INV_SUCCESS or a non-zero error code.

5.10.2.3 unsigned short inv_get_accel_id (void)

Get the ID of the accel in use.

Returns:

ID of the accel in use.

5.10.2.4 unsigned char inv_get_slave_addr (void)

Query the accel slave address.

Returns:

The 7-bit accel slave address.

5.11 COMPASSDL

Motion Library - Compass Driver Layer.

Files

- file ak8972.c

  Magnetometer setup and handling methods for the AKM AK8972 compass device.

- file ak8975.c

  Magnetometer setup and handling methods for the AKM AK8975, AKM AK8975B, and AKM AK8975C compass devices.

- file ami306.c

  Magnetometer setup and handling methods for Aichi AMI306 compass.

- file ami30x.c

  Magnetometer setup and handling methods for Aichi AMI304 and AMI305 compass devices.

- file compass.c

  Compass setup and handling methods.

- file hmc5883.c

  Magnetometer setup and handling methods for Honeywell HMC5883 compass.

- file hscdtd002b.c

  Magnetometer setup and handling methods for Alps HSCDTD002B compass.

- file hscdtd004a.c

  Magnetometer setup and handling methods for Alps HSCDTD004A compass.

- file lsm303dlx_m.c

  Magnetometer setup and handling methods for ST LSM303 compass.

- file mmc314x.c

  Magnetometer setup and handling methods for the MEMSIC MMC314x compass.

- file yas529.c

  Magnetometer setup and handling methods for Yamaha YAS529 compass when used in a user-space solution (no kernel driver).

- file yas530-kernel.c

  Magnetometer setup and handling methods for Yamaha YAS530 compass when interfacing with the kernel.

- file yas530.c

  Magnetometer setup and handling methods for Yamaha YAS530 compass when used in a user-space solution (no kernel driver).

Functions

- unsigned char inv_compass_present (void)

  Used to determine if a compass is configured and used by the MPL.

- inv_error_t inv_compass_read_reg (unsigned char reg, unsigned char ∗val)

  Read values from the compass slave device registers, regardless of the bus it is connected to and the MPU's configuration.

- inv_error_t inv_compass_read_scale (long ∗val)

  Read values from the compass slave device scale registers, regardless of the bus it is connected to and the MPU's configuration.

- inv_error_t inv_compass_write_reg (unsigned char reg, unsigned char val)

  Write a single register on the compass slave device, regardless of the bus it is connected to and the MPU's configuration.

- inv_error_t inv_get_compass_data (long ∗data)

  Get a sample of compass data from the device.

- unsigned short inv_get_compass_id (void)

  Get the ID of the compass in use.

- unsigned char inv_get_compass_slave_addr (void)

  Query the compass slave address.

- inv_error_t inv_set_compass_bias (struct compass_obj_t ∗obj, long ∗bias)

  Sets the compass bias.

5.11.1 Detailed Description

Motion Library - Compass Driver Layer.

Provides the interface to setup and handle an compass connected to either the primary or the seconday I2C interface of the gyroscope.

5.11.2 Function Documentation

5.11.2.1 unsigned char inv_compass_present (void)

Used to determine if a compass is configured and used by the MPL.

Returns:

INV_SUCCESS if the compass is present.

5.11.2.2 inv_error_t inv_compass_read_reg (unsigned char *reg*, unsigned char * *val*)

Read values from the compass slave device registers, regardless of the bus it is connected to and the MPU's configuration.

Parameters:

reg the register to read from on the slave compass device.

val a buffer of 3 elements to store the values read from the compass device.

Returns:

INV_SUCCESS = 0 if successful. A non-zero error code otherwise.

5.11.2.3 inv_error_t inv_compass_read_scale (long * *val*)

Read values from the compass slave device scale registers, regardless of the bus it is connected to and the MPU's configuration.

Parameters:

reg the register to read from on the slave compass device.

val a buffer of 3 elements to store the values read from the compass device.

Returns:

INV_SUCCESS = 0 if successful. A non-zero error code otherwise.

5.11.2.4 inv_error_t inv_compass_write_reg (unsigned char *reg*, unsigned char *val*)

Write a single register on the compass slave device, regardless of the bus it is connected to and the MPU's configuration.

Parameters:

reg the register to write to on the slave compass device.

val the value to write.

Returns:

INV_SUCCESS = 0 if successful. A non-zero error code otherwise.

5.11.2.5 inv_error_t inv_get_compass_data (long * *data*)

Get a sample of compass data from the device.

Parameters:

data the buffer to store the compass raw data for X, Y, and Z axes.

Returns:

INV_SUCCESS or a non-zero error code.

5.11.2.6 unsigned short inv_get_compass_id (void)

Get the ID of the compass in use.

Returns:

ID of the compass in use.

5.11.2.7 unsigned char inv_get_compass_slave_addr (void)

Query the compass slave address.

Returns:

The 7-bit compass slave address.

5.11.2.8 inv_error_t inv_set_compass_bias (struct compass_obj_t * *obj*, long * *bias*)

Sets the compass bias.

Parameters:

bias Compass bias, length 3. Scale is chip units * 2^16. Frame is mount frame which may be different from body frame.

Returns:

INV_SUCCESS = 0 if successful. A non-zero error code otherwise.

5.12 TEMP_COMP

Gyroscope learning temperature compensation.

Functions

- inv_error_t inv_disable_temp_comp (void)

  Disable the temperature compensation algorithm and calibrated gyro temperature compensated output.

- inv_error_t inv_enable_temp_comp (void)

  Enable the temperature compensation algorithm and calibrated gyro temperature compensated output.

- float inv_get_calibration_temp_difference (void)

  Get the temperature change from last and previous time temp_comp_apply has executed.

- inv_error_t inv_get_gyro_temp_slope (long *data)

  inv_get_gyro_temp_slope is used to get the temperature compensation algorithm's estimate of the gyroscope bias temperature coefficient.

- inv_error_t inv_get_gyro_temp_slope_float (float *data)

  inv_get_gyro_temp_slope_float is used to get the temperature compensation algorithm's estimate of the gyroscope bias temperature coefficient.

- inv_error_t inv_set_dmp_slope (float slope_x, float slope_y, float slope_z)

  Gyro slope in dps which gets pushed down to DMP for temperature correction on the DMP.

- inv_error_t inv_set_gyro_temp_slope (long *data)

  inv_set_gyro_temp_slope is used to set the temperature compensation algorithm's estimate of the gyroscope bias temperature coefficient.

- inv_error_t inv_set_gyro_temp_slope_float (float *data)

  inv_set_gyro_temp_slope_float is used to get the temperature compensation algorithm's estimate of the gyroscope bias temperature coefficient.

- int inv_temp_comp_find_bin (float temp)

  Find the right temperature bin.

- int inv_temp_comp_has_slope (void)

  Whether the temperature compensation slope has been already computed.

- inv_error_t inv_temp_comp_is_enabled (unsigned char *is_enabled)

  inv_temp_comp_is_enabled *checks if the temperature compensation algorithm is being used to update gyro biases.*

- inv_error_t inv_temp_comp_reset (unsigned char new_state)

  Reset the temperature compensation algorithm internal state machine.

- inv_error_t inv_temp_comp_supervisor (struct inv_obj_t *inv_obj)

  Main entry point of the temperature compensation algorithm.

- inv_error_t temp_comp_load_calibration_handler (void)

  Apply temperature compensation table to gyro bias.

- void temp_comp_print_table (void)

  Prints the temperature compesation table for debugging purpose.

5.12.1 Detailed Description

Gyroscope learning temperature compensation.

5.12.2 Function Documentation

5.12.2.1 inv_error_t inv_disable_temp_comp (void)

Disable the temperature compensation algorithm and calibrated gyro temperature compensated output.

Returns:

INV_SUCCESS == 0.

5.12.2.2 inv_error_t inv_enable_temp_comp (void)

Enable the temperature compensation algorithm and calibrated gyro temperature compensated output.

Returns:

INV_SUCCESS == 0.

5.12.2.3 float inv_get_calibration_temp_difference (void)

Get the temperature change from last and previous time temp_comp_apply has executed.

This is a sinple shorthand to avoid making the internal data structure struct _TC non file static.

Returns:

The temperature difference in degrees C.

5.12.2.4 inv_error_t inv_get_gyro_temp_slope (long * *data*)

inv_get_gyro_temp_slope is used to get the temperature compensation algorithm's estimate of the gyroscope bias temperature coefficient.

The argument array elements are ordered X,Y,Z. Values are in units of dps per deg C (degrees per second per degree Celcius). Values are scaled so that 1 dps per deg C = 2^16 LSBs. Please refer to the provided "9-Axis Sensor Fusion Application Note" document.

Precondition:

MLDmpOpen() or MLDmpPedometerStandAloneOpen() must have been called.

Parameters:

data A pointer to an array to be passed back to the user. **Must be 3 cells long** .

Returns:

Zero if the command is successful; an ML error code otherwise.

5.12.2.5 inv_error_t inv_get_gyro_temp_slope_float (float * *data*)

inv_get_gyro_temp_slope_float is used to get the temperature compensation algorithm's estimate of the gyroscope bias temperature coefficient.

The argument array elements are ordered X,Y,Z. Values are in units of dps per deg C (degrees per second per degree Celcius) Please refer to the provided "9-Axis Sensor Fusion Application Note" document.

Precondition:

MLDmpOpen() or MLDmpPedometerStandAloneOpen() must have been called.

Parameters:

data A pointer to an array to be passed back to the user. **Must be 3 cells long** .

Returns:

INV_SUCCESS if the command is successful; an error code otherwise.

5.12.2.6 inv_error_t inv_set_gyro_temp_slope (long * *data*)

inv_set_gyro_temp_slope is used to set the temperature compensation algorithm's estimate of the gyroscope bias temperature coefficient.

The argument array elements are ordered X,Y,Z. Values are in units of dps per deg C (degrees per second per degree Celcius), and scaled such that 1 dps per deg C = 2^16 LSBs. Please refer to the provided "9-Axis Sensor Fusion Application Note" document.

inv_set_gyro_temp_slope is used to set Gyro temperature slope

Precondition:

MLDmpOpen() or MLDmpPedometerStandAloneOpen()

Parameters:

data A pointer to an array to be copied from the user.

Returns:

INV_SUCCESS if successful; a non-zero error code otherwise.

5.12.2.7 inv_error_t inv_set_gyro_temp_slope_float (float * *data*)

inv_set_gyro_temp_slope_float is used to get the temperature compensation algorithm's estimate of the gyroscope bias temperature coefficient.

The argument array elements are ordered X,Y,Z. Values are in units of dps per deg C (degrees per second per degree Celcius)

Please refer to the provided "9-Axis Sensor Fusion Application Note" document provided.

Precondition:

MLDmpOpen() or MLDmpPedometerStandAloneOpen()

Parameters:

data A pointer to an array to be copied from the user.

Returns:

INV_SUCCESS if successful; a non-zero error code otherwise.

5.12.2.8 int inv_temp_comp_find_bin (float *temp*)

Find the right temperature bin.

Parameters:

temp The temperature in degree C.

Returns:

the temperature bin number, [0, BINS).

5.12.2.9 inv_error_t inv_temp_comp_is_enabled (unsigned char * *is_enabled*)

inv_temp_comp_is_enabled checks if the temperature compensation algorithm is being used to update gyro biases.

Parameters:

is_enabled True if temp comp is enabled.

Returns:

INV_SUCCESS if successful.

5.12.2.10 inv_error_t inv_temp_comp_supervisor (struct inv_obj_t * *inv_obj*)

Main entry point of the temperature compensation algorithm.

Parameters:

inv_obj A pointer to the internal struct inv_obj_t data structure.

Returns:

INV_SUCCESS == 0.

5.12.2.11 inv_error_t temp_comp_load_calibration_handler (void)

Apply temperature compensation table to gyro bias.

Recompute the temperature compensation table using the values loaded from file. Apply the value by finding the best guess for the gyro offsets.

Returns:

INV_SUCCESS if successful, a non-zero error code otherwise.

5.13 ML_STORED_DATA

Files

- file ml_stored_data.c

  functions for reading and writing stored data sets.

Functions

- int FindTempBin (float temp)

  Duplicate of the inv_temp_comp_find_bin function in the libmpl advanced algorithms library.

- unsigned int inv_get_cal_length (void)

  Returns the length of the **MPL internal calibration data**.

- inv_error_t inv_load_cal (unsigned char *calData)

  Loads a set of calibration data.

- inv_error_t inv_load_cal_V0 (unsigned char *calData, unsigned short len)

  Loads a type 0 set of calibration data.

- inv_error_t inv_load_cal_V1 (unsigned char *calData, unsigned short len)

  Loads a type 1 set of calibration data.

- inv_error_t inv_load_cal_V2 (unsigned char *calData, unsigned short len)

  Loads a type 2 set of calibration data.

- inv_error_t inv_load_cal_V3 (unsigned char *calData, unsigned short len)

  Loads a type 3 set of calibration data.

- inv_error_t inv_load_cal_V4 (unsigned char *calData, unsigned short len)

  Loads a type 4 set of calibration data.

- inv_error_t inv_load_cal_V5 (unsigned char *calData, unsigned short len)

  Loads a type 5 set of calibration data.

- inv_error_t inv_load_calibration (void)

  Load a calibration file.

- inv_error_t inv_store_cal (unsigned char *calData, int length)

Stores a set of calibration data.

- inv_error_t inv_store_calibration (void)
  Store runtime calibration data to a file.

5.13.1 Function Documentation

5.13.1.1 int FindTempBin (float *temp*)

Duplicate of the inv_temp_comp_find_bin function in the libmpl advanced algorithms library.

To remove cross-dependency, for now, we reimplement the same function here.

Parameters:

temp the temperature (1 count == 1 degree C).

5.13.1.2 unsigned int inv_get_cal_length (void)

Returns the length of the **MPL internal calibration data**.

Should be called before allocating the memory required to store this data to a file. This function returns the total size required to store the cal data including the header (4 bytes) and the checksum (2 bytes).

Precondition:

Must be in INV_STATE_DMP_OPENED state. inv_dmp_open() or inv_dmp_stop() must have been called. inv_dmp_start() and inv_dmp_close() must have **NOT** been called.

Returns:

the length of the internal calibrated data format.

5.13.1.3 inv_error_t inv_load_cal (unsigned char * *calData*)

Loads a set of calibration data.

It parses a binary data set containing calibration data. The binary data set is intended to be loaded from a file.

Precondition:

inv_dmp_open()

or inv_open_low_power_pedometer() or inv_eis_open_dmp()

must have been called.

Parameters:

calData A pointer to an array of bytes to be parsed.

Returns:

INV_SUCCESS if successful, a non-zero error code otherwise.

5.13.1.4 inv_error_t inv_load_cal_V0 (unsigned char * *calData*, unsigned short *len*)

Loads a type 0 set of calibration data.

It parses a binary data set containing calibration data. The binary data set is intended to be loaded from a file. This calibrations data format stores values for (in order of appearance) :

- temperature,
- gyro biases for X, Y, Z axes. This calibration data would normally be produced by the MPU Self Test and its size is 18 bytes (header and checksum included). Calibration format type 0 is currently **NOT** used and is substituted by type 5 : inv_load_cal_V5().

Note:

This calibration data format is obsoleted and no longer supported by the rest of the MPL

Precondition:

inv_dmp_open()

or inv_open_low_power_pedometer() or inv_eis_open_dmp()

must have been called.

Parameters:

calData A pointer to an array of bytes to be parsed.

len the length of the calibration

Returns:

INV_SUCCESS if successful, a non-zero error code otherwise.

5.13.1.5 inv_error_t inv_load_cal_V1 (unsigned char * *calData*, unsigned short *len*)

Loads a type 1 set of calibration data.

It parses a binary data set containing calibration data. The binary data set is intended to be loaded from a file. This calibrations data format stores values for (in order of appearance) :

- temperature,
- gyro biases for X, Y, Z axes,
- accel biases for X, Y, Z axes. This calibration data would normally be produced by the MPU Self Test and its size is 24 bytes (header and checksum included). Calibration format type 1 is currently **NOT** used and is substituted by type 5 : inv_load_cal_V5().

Note:

In order to successfully work, the gyro bias must be stored expressed in 250 dps full scale (131.072 sensitivity). Other full scale range will produce unpredictable results in the gyro biases.

Precondition:

inv_dmp_open()

or inv_open_low_power_pedometer() or inv_eis_open_dmp()

must have been called.

Parameters:

calData A pointer to an array of bytes to be parsed.

len the length of the calibration

Returns:

INV_SUCCESS if successful, a non-zero error code otherwise.

5.13.1.6 inv_error_t inv_load_cal_V2 (unsigned char * *calData*, unsigned short *len*)

Loads a type 2 set of calibration data.

It parses a binary data set containing calibration data. The binary data set is intended to be loaded from a file. This calibrations data format stores values for (in order of appearance) :

- temperature compensation : temperature data points,
- temperature compensation : gyro biases data points for X, Y, and Z axes.
- accel biases for X, Y, Z axes. This calibration data is produced internally by the MPL and its size is 2222 bytes (header and checksum included). Calibration format type 2 is currently **NOT** used and is substituted by type 4 : inv_load_cal_V4().

Precondition:

inv_dmp_open()

or inv_open_low_power_pedometer() or inv_eis_open_dmp()

must have been called.

Parameters:

calData A pointer to an array of bytes to be parsed.

len the length of the calibration

Returns:

INV_SUCCESS if successful, a non-zero error code otherwise.

5.13.1.7 inv_error_t inv_load_cal_V3 (unsigned char * *calData*, unsigned short *len*)

Loads a type 3 set of calibration data.

It parses a binary data set containing calibration data. The binary data set is intended to be loaded from a file. This calibrations data format stores values for (in order of appearance) :

- temperature compensation : temperature data points,
- temperature compensation : gyro biases data points for X, Y, and Z axes.
- accel biases for X, Y, Z axes.
- compass biases for X, Y, Z axes and bias tracking algorithm mock-up. This calibration data is produced internally by the MPL and its size is 2429 bytes (header and checksum included). Calibration format type 3 is currently **NOT** used and is substituted by type 4 : inv_load_cal_V4().

Precondition:

inv_dmp_open()

or inv_open_low_power_pedometer() or inv_eis_open_dmp()

must have been called.

Parameters:

calData A pointer to an array of bytes to be parsed.

len the length of the calibration

Returns:

INV_SUCCESS if successful, a non-zero error code otherwise.

5.13.1.8 inv_error_t inv_load_cal_V4 (unsigned char * *calData*, unsigned short *len*)

Loads a type 4 set of calibration data.

It parses a binary data set containing calibration data. The binary data set is intended to be loaded from a file. This calibrations data format stores values for (in order of appearance) :

- temperature compensation : temperature data points,
- temperature compensation : gyro biases data points for X, Y, and Z axes.
- accel biases for X, Y, Z axes.
- compass biases for X, Y, Z axes, compass scale, and bias tracking algorithm mock-up. This calibration data is produced internally by the MPL and its size is 2777 bytes (header and checksum included). Calibration format type 4 is currently used and substitutes type 2 (inv_load_cal_V2()) and 3 (inv_load_cal_-V3()).

Precondition:

inv_dmp_open()

or inv_open_low_power_pedometer() or inv_eis_open_dmp()

must have been called.

Parameters:

calData A pointer to an array of bytes to be parsed.

len the length of the calibration

Returns:

INV_SUCCESS if successful, a non-zero error code otherwise.

5.13.1.9 inv_error_t inv_load_cal_V5 (unsigned char * *calData*, unsigned short *len*)

Loads a type 5 set of calibration data.

It parses a binary data set containing calibration data. The binary data set is intended to be loaded from a file. This calibrations data format stores values for (in order of appearance) :

- temperature,
- gyro biases for X, Y, Z axes,
- accel biases for X, Y, Z axes. This calibration data would normally be produced by the MPU Self Test and its size is 36 bytes (header and checksum included). Calibration format type 5 is produced by the MPU Self Test and substitutes the type 1 : inv_load_cal_V1().

Precondition:

inv_dmp_open()

or inv_open_low_power_pedometer() or inv_eis_open_dmp()

must have been called.

Parameters:

calData A pointer to an array of bytes to be parsed.

len the length of the calibration

Returns:

INV_SUCCESS if successful, a non-zero error code otherwise.

5.13.1.10 inv_error_t inv_load_calibration (void)

Load a calibration file.

Precondition:

Must be in INV_STATE_DMP_OPENED state. inv_dmp_open() or inv_dmp_stop() must have been called. inv_dmp_start() and inv_dmp_close() must have **NOT** been called.

Returns:

0 or error code.

5.13.1.11 inv_error_t inv_store_cal (unsigned char * *calData*, int *length*)

Stores a set of calibration data.

It generates a binary data set containing calibration data. The binary data set is intended to be stored into a file.

Precondition:

inv_dmp_open()

Parameters:

calData A pointer to an array of bytes to be stored.

length The amount of bytes available in the array.

Returns:

INV_SUCCESS if successful, a non-zero error code otherwise.

5.13.1.12 inv_error_t inv_store_calibration (void)

Store runtime calibration data to a file.

Precondition:

Must be in INV_STATE_DMP_OPENED state. inv_dmp_open() or inv_dmp_stop() must have been called. inv_dmp_start() and inv_dmp_close() must have **NOT** been called.

Returns:

0 or error code.

5.14 MPU_SELF_TEST

C wrapper to integrate the MPU Self Test wrapper in MPL.

Files

- file ml_mputest.c

  C wrapper to integrate the MPU Self Test wrapper in MPL.

- file mputest.c

  MPU Self Test routines for assessing gyro sensor status after surface mount has happened on the target host platform.

Functions

- int inv_device_test (void *mlsl_handle, uint_fast8_t sensor_mask, uint_fast8_t perform_full_test, uint_fast8_t provide_result)

  The main entry point of the MPU Self Test, triggering the run of the single tests, for gyros and accelerometers.

- inv_error_t inv_self_test_accel_z_run (void)

  Runs the Accelerometer Calibration Test at MPL runtime.

- inv_error_t inv_self_test_bias_run (void)

  Runs the MPU test for bias correction only at MPL runtime (the short version is run but test results are ignored).

- inv_error_t inv_self_test_calibration_run (void)

  Runs the MPU Calibration Test at MPL runtime (long version).

- inv_error_t inv_self_test_run (void)

  Runs the MPU Self Test at MPL runtime (short version).

- inv_error_t inv_self_test_set_accel_z_orient (signed char z_sign)

  Set the orientation of the acceleroemter Z axis as it will be expected when running the MPU Self Test.

- void inv_set_test_parameters (unsigned int slave_addr, float sensitivity, int p_thresh, float total_time_tol, int bias_thresh, unsigned short accel_samples)

  Modify the self test limits from their default values.

- int inv_test_accel (void *mlsl_handle, int enable_axes, short *bias, long gravity, uint_fast8_t perform_full_test)

  If requested via inv_test_setup_accel(), test the accelerometer biases and calculate the necessary bias correction.

- int inv_test_gyro (void *mlsl_handle, short gyro_biases[3], short *temp_avg, uint_fast8_t perform_full_test)

  Test the gyroscope sensor.

5.14.1 Detailed Description

C wrapper to integrate the MPU Self Test wrapper in MPL.

MPU Self Test functions.

Provides ML name compliant naming and an additional API that automates the suspension of normal MPL operations, runs the test, and resume.

These functions provide an in-site test of the MPU chips. The main entry point is the inv_mpu_test function. This runs the tests (as described in the accompanying documentation) and writes a configuration file containing initial calibration data. inv_mpu_test returns INV_SUCCESS if the chip passes the tests. Otherwise, an error code is returned. The functions in this file rely on MLSL and MLOS: refer to the MPL documentation for more information regarding the system interface files.

5.14.2 Function Documentation

5.14.2.1 int inv_device_test (void * *mlsl_handle*, uint_fast8_t *sensor_mask*, uint_fast8_t *perform_full_test*, uint_fast8_t *provide_result*)

The main entry point of the MPU Self Test, triggering the run of the single tests, for gyros and accelerometers.

Prepares the MPU for the test, taking the device out of low power state if necessary, switching the MPU secondary I2C interface into bypass mode and restoring the original power state at the end of the test. This function is also responsible for encoding the output of each test in the correct format as it is stored on the file/medium of choice (according to inv_serial_write_cal() function). The format needs to stay perfectly consistent with the one expected by the corresponding loader in ml_stored_data.c; currectly the loaded in use is inv_load_cal_V1 (record type 1 - initial calibration).

Parameters:

mlsl_handle serial interface handle to allow serial communication with the device, both gyro and accelerometer.

perform_full_test If 1: Complete calibration test: Calculate offset, drive frequency, and noise and compare it against set thresholds. When 0: Skip the noise and drive frequency calculation, simply calculate the gyro biases.

provide_result If 1: Report the final result using a bit-mask like error code as described in the inv_test_gyro() function.

Returns:

0 on success. A non-zero error code on error. Propagates the errors from the tests up to the caller.

5.14.2.2 inv_error_t inv_self_test_accel_z_run (void)

Runs the Accelerometer Calibration Test at MPL runtime.

If the DMP is operating, stops the DMP temporarely, runs the Accel Calibration Test, and re-starts the DMP.

Returns:

INV_SUCCESS or a non-zero error code otherwise.

5.14.2.3 inv_error_t inv_self_test_bias_run (void)

Runs the MPU test for bias correction only at MPL runtime (the short version is run but test results are ignored).

If the DMP is operating, stops the DMP temporarely, runs the bias calculation routines, and re-starts the DMP.

Returns:

INV_SUCCESS or a non-zero error code otherwise.

5.14.2.4 inv_error_t inv_self_test_calibration_run (void)

Runs the MPU Calibration Test at MPL runtime (long version).

If the DMP is operating, stops the DMP temporarely, runs the MPU Calibration Test, and re-starts the DMP.

Returns:

INV_SUCCESS or a non-zero error code otherwise.

5.14.2.5 inv_error_t inv_self_test_run (void)

Runs the MPU Self Test at MPL runtime (short version).

If the DMP is operating, stops the DMP temporarely, runs the MPU Self Test, and re-starts the DMP.

Returns:

INV_SUCCESS or a non-zero error code otherwise.

5.14.2.6 inv_error_t inv_self_test_set_accel_z_orient (signed char *z_sign*)

Set the orientation of the acceleroemter Z axis as it will be expected when running the MPU Self Test.

Specifies the orientation of the accelerometer Z axis : Z axis pointing upwards or downwards.

Parameters:

z_sign The sign of the accelerometer Z axis; valid values are +1 and -1 for +Z and -Z respectively. Any other value will cause the setting to be ignored and an error code to be returned. Note that this setting is an override on top of the chip mounting matrix in use and its purpose is to allow the accel test to be run with the -Z axis facing up instead of +Z axis.

Returns:

INV_SUCCESS or a non-zero error code.

5.14.2.7 void inv_set_test_parameters (unsigned int *slave_addr*, float *sensitivity*, int *p_thresh*, float *total_time_tol*, int *bias_thresh*, unsigned short *accel_samples*)

Modify the self test limits from their default values.

Parameters:

slave_addr the slave address the MPU device is setup to respond at. The default is DEF_MPU_ADDR = 0x68.

sensitivity the read sensitivity of the device in LSB/dps as it is trimmed. NOTE : if using the self test as part of the MPL, the sensitivity the different sensitivity trims are already taken care of.

p_thresh number of packets expected to be received in one test period. Depends on the sampling frequency of choice (set by default to 1 kHz) and low pass filter cut-off frequency selection (set to 42 Hz). The default is DEF_PACKET_THRESH = 75 packets.

total_time_tol time skew tolerance, taking into account imprecision in turning the FIFO on and off and the processor time imprecision (for 1 GHz processor). The default is DEF_TOTAL_TIMING_TOL = 3 %, about 2 packets for a 75ms period.

bias_thresh bias level threshold, the maximun acceptable no motion bias for a production quality part. The default is DEF_BIAS_THRESH = 40 dps.

accel_samples the number of samples to be collected from the accelerometer device to estimate its initial biases.

5.14.2.8 int inv_test_accel (void * *mlsl_handle*, int *enable_axes*, short * *bias*, long *gravity*, uint_fast8_t *perform_full_test*)

If requested via inv_test_setup_accel(), test the accelerometer biases and calculate the necessary bias correction.

Parameters:

mlsl_handle serial interface handle to allow serial communication with the device, both gyro and accelerometer.

enable_axis specify which axis has to be checked and corrected: provides a switch mode between 3 axis calibration and Z axis only calibration.

bias output pointer to store the initial bias calculation provided by the MPU Self Test. Requires 3 elements to store accel X, Y, and Z axis bias.

gravity The gravity value given the parts' sensitivity: for example if the accelerometer is set to +/- 2 gee ==> the gravity value will be 2^14 = 16384.

perform_full_test If 1: calculates offsets and noise and compare it against set thresholds. The final exist status will reflect if any of the value is outside of the expected range. When 0; skip the noise calculation and pass/fail assessment; simply calculates the accel biases.

Returns:

0 on success. A non-zero error code on error.

5.14.2.9 int inv_test_gyro (void * *mlsl_handle*, short *gyro_biases*[3], short * *temp_avg*, uint_fast8_t *perform_full_test*)

Test the gyroscope sensor.

Implements the core logic of the MPU Self Test. Produces the PASS/FAIL result. Loads the calculated gyro biases and temperature datum into the corresponding pointers.

Parameters:

mlsl_handle serial interface handle to allow serial communication with the device, both gyro and accelerometer.

gyro_biases output pointer to store the initial bias calculation provided by the MPU Self Test. Requires 3 elements for gyro X, Y, and Z.

temp_avg output pointer to store the initial average temperature as provided by the MPU Self Test.

perform_full_test If 1: Complete calibration test: Calculate offset, drive frequency, and noise and compare it against set thresholds. When 0: Skip the noise and drive frequency calculation, simply calculate the gyro biases.

Returns:

0 on success. On error, the return value is a bitmask representing: 0, 1, 2 Failures with PLLs on X, Y, Z gyros respectively (decimal values will be 1, 2, 4 respectively). 3, 4, 5 Excessive offset with X, Y, Z gyros respectively (decimal values will be 8, 16, 32 respectively). 6, 7, 8 Excessive noise with X, Y, Z gyros respectively (decimal values will be 64, 128, 256 respectively). 9 If any of the RMS noise values is zero, it may be due to a non-functional gyro or FIFO/register failure. (decimal value will be 512).

5.15 MLSL

Motion Library - Serial Layer.

Files

- file mlsl.h

  The Motion Library System Layer.

Functions

- inv_error_t inv_serial_close (void *sl_handle)

  inv_serial_close() - used to close the serial port.

- inv_error_t inv_serial_get_cal_length (unsigned int *len)

  inv_serial_get_cal_length() - Get the calibration length from the storage.

- inv_error_t inv_serial_open (char const *port, void **sl_handle)

  inv_serial_open() - used to open the serial port.

- inv_error_t inv_serial_read (void *sl_handle, unsigned char slave_addr, unsigned char register_addr, unsigned short length, unsigned char *data)

  inv_serial_read() - used to read multiple bytes of data from registers.

- inv_error_t inv_serial_read_cal (unsigned char *cal, unsigned int len)

  inv_serial_read_cal() - used to get the calibration data.

- inv_error_t inv_serial_read_cfg (unsigned char *cfg, unsigned int len)

  inv_serial_read_cfg() - used to get the configuration data.

- inv_error_t inv_serial_read_fifo (void *sl_handle, unsigned char slave_addr, unsigned short length, unsigned char *data)

  inv_serial_read_fifo() - used to read multiple bytes of data from the fifo.

- inv_error_t inv_serial_read_mem (void *sl_handle, unsigned char slave_addr, unsigned short mem_addr, unsigned short length, unsigned char *data)

  inv_serial_read_mem() - used to read multiple bytes of data from the memory.

- inv_error_t inv_serial_reset (void *sl_handle)

  inv_serial_reset() - used to reset any buffering the driver may be doing returns INV_SUCCESS if successful, a non-zero error code otherwise.

- inv_error_t inv_serial_single_write (void *sl_handle, unsigned char slave_addr, unsigned char register_addr, unsigned char data)

  inv_serial_single_write() - used to write a single byte of data.

- inv_error_t inv_serial_write (void *sl_handle, unsigned char slave_addr, unsigned short length, unsigned char const *data)

  inv_serial_write() - used to write multiple bytes of data to registers.

- inv_error_t inv_serial_write_cal (unsigned char *cal, unsigned int len)

  inv_serial_write_cal() - used to save the calibration data.

- inv_error_t inv_serial_write_cfg (unsigned char *cfg, unsigned int len)

  inv_serial_write_cfg() - used to save the configuration data.

- inv_error_t inv_serial_write_fifo (void *sl_handle, unsigned char slave_addr, unsigned short length, unsigned char const *data)

  inv_serial_write_fifo() - used to write multiple bytes of data to the fifo.

- inv_error_t inv_serial_write_mem (void *sl_handle, unsigned char slave_addr, unsigned short mem_addr, unsigned short length, unsigned char const *data)

  inv_serial_write_mem() - used to write multiple bytes of data to the memory.

5.15.1 Detailed Description

Motion Library - Serial Layer.

The Motion Library System Layer provides the Motion Library with the communication interface to the hardware.

The communication interface is assumed to support serial transfers in burst of variable length up to SERIAL_MAX_TRANSFER_SIZE. The default value for SERIAL_MAX_TRANSFER_SIZE is 128 bytes. Transfers of length greater than SERIAL_MAX_TRANSFER_SIZE, will be subdivided in smaller transfers of length <= SERIAL_MAX_TRANSFER_SIZE. The SERIAL_MAX_TRANSFER_SIZE definition can be modified to overcome any host processor transfer size limitation down to 1 B, the minimum. An higher value for SERIAL_MAX_TRANSFER_SIZE will favor performance and efficiency while requiring higher resource usage (mostly buffering). A smaller value will increase overhead and decrease efficiency but allows to operate with more resource constrained processor and master serial controllers. The SERIAL_MAX_TRANSFER_SIZE definition can be found in the mlsl.h header file and master serial controllers. The SERIAL_MAX_TRANSFER_SIZE definition can be found in the mlsl.h header file.

5.15.2 Function Documentation

5.15.2.1 inv_error_t inv_serial_close (void * *sl_handle*)

inv_serial_close() - used to close the serial port.

a file handle to the serial device used for the communication.

This port is used to send and receive data to the device.

This function is called by inv_serial_stop(). Unlike previous MPL Software releases, explicitly calling inv_serial_stop() is mandatory to properly shut-down the communication with the device.

returns INV_SUCCESS if successful, a non-zero error code otherwise.

5.15.2.2 inv_error_t inv_serial_get_cal_length (unsigned int * *len*)

inv_serial_get_cal_length() - Get the calibration length from the storage.

lenght to be returned

returns INV_SUCCESS if successful, a non-zero error code otherwise.

5.15.2.3 inv_error_t inv_serial_open (char const * *port*, void ** *sl_handle*)

inv_serial_open() - used to open the serial port.

The COM port specification associated with the device in use. a pointer to the file handle to the serial device to be open for the communication. This port is used to send and receive data to the device.

This function is called by inv_serial_start(). Unlike previous MPL Software releases, explicitly calling inv_serial_start() is mandatory to instantiate the communication with the device.

returns INV_SUCCESS if successful, a non-zero error code otherwise.

5.15.2.4 inv_error_t inv_serial_read (void * *sl_handle*, unsigned char *slave_addr*, unsigned char *register_addr*, unsigned short *length*, unsigned char * *data*)

inv_serial_read() - used to read multiple bytes of data from registers.

a file handle to the serial device used for the communication. I2C slave address of device. Register address to read. Length of burst of data. Pointer to block of data.

returns INV_SUCCESS == 0 if successful; a non-zero error code otherwise.

5.15.2.5 inv_error_t inv_serial_read_cal (unsigned char * *cal*, unsigned int *len*)

inv_serial_read_cal() - used to get the calibration data.

Pointer to the calibration data. Length of the calibration data.

It is called by the MPL to get the calibration data used by the motion library. This data is typically be saved in non-volatile memory.

returns INV_SUCCESS if successful, a non-zero error code otherwise.

5.15.2.6 inv_error_t inv_serial_read_cfg (unsigned char * *cfg*, unsigned int *len*)

inv_serial_read_cfg() - used to get the configuration data.

Pointer to the configuration data. Length of the configuration data.

Is called by the MPL to get the configuration data used by the motion library. This data would typically be saved in non-volatile memory.

returns INV_SUCCESS if successful, a non-zero error code otherwise.

5.15.2.7 inv_error_t inv_serial_read_fifo (void * *sl_handle*, unsigned char *slave_addr*, unsigned short *length*, unsigned char * *data*)

inv_serial_read_fifo() - used to read multiple bytes of data from the fifo.

a file handle to the serial device used for the communication. I2C slave address of device. Length of burst of data. Pointer to block of data.

returns INV_SUCCESS == 0 if successful; a non-zero error code otherwise.

5.15.2.8 inv_error_t inv_serial_read_mem (void * *sl_handle*, unsigned char *slave_addr*, unsigned short *mem_addr*, unsigned short *length*, unsigned char * *data*)

inv_serial_read_mem() - used to read multiple bytes of data from the memory.

This should be sent by I2C or SPI.

a file handle to the serial device used for the communication. I2C slave address of device. The location in the memory to read from. Length of burst data. Pointer to block of data.

returns INV_SUCCESS == 0 if successful; a non-zero error code otherwise.

5.15.2.9 inv_error_t inv_serial_single_write (void * *sl_handle*, unsigned char *slave_addr*, unsigned char *register_addr*, unsigned char *data*)

inv_serial_single_write() - used to write a single byte of data.

pointer to the serial device used for the communication. I2C slave address of device. Register address to write. Single byte of data to write.

It is called by the MPL to write a single byte of data to the MPU.

returns INV_SUCCESS if successful, a non-zero error code otherwise.

5.15.2.10 inv_error_t inv_serial_write (void * *sl_handle*, unsigned char *slave_addr*, unsigned short *length*, unsigned char const * *data*)

inv_serial_write() - used to write multiple bytes of data to registers.

a file handle to the serial device used for the communication. I2C slave address of device. Register address to write. Length of burst of data. Pointer to block of data.

returns INV_SUCCESS if successful, a non-zero error code otherwise.

5.15.2.11 inv_error_t inv_serial_write_cal (unsigned char * *cal*, unsigned int *len*)

inv_serial_write_cal() - used to save the calibration data.

Pointer to the calibration data. Length of the calibration data.

It is called by the MPL to save the calibration data used by the motion library. This data is typically be saved in non-volatile memory. returns INV_SUCCESS if successful, a non-zero error code otherwise.

5.15.2.12 inv_error_t inv_serial_write_cfg (unsigned char * *cfg*, unsigned int *len*)

inv_serial_write_cfg() - used to save the configuration data.

Pointer to the configuration data. Length of the configuration data.

Is called by the MPL to save the configuration data used by the motion library. This data would typically be saved in non-volatile memory.

returns INV_SUCCESS if successful, a non-zero error code otherwise.

5.15.2.13 inv_error_t inv_serial_write_fifo (void * *sl_handle*, unsigned char *slave_addr*, unsigned short *length*, unsigned char const * *data*)

inv_serial_write_fifo() - used to write multiple bytes of data to the fifo.

a file handle to the serial device used for the communication. I2C slave address of device. Length of burst of data. Pointer to block of data.

returns INV_SUCCESS == 0 if successful; a non-zero error code otherwise.

5.15.2.14 inv_error_t inv_serial_write_mem (void * *sl_handle*, unsigned char *slave_addr*, unsigned short *mem_addr*, unsigned short *length*, unsigned char const * *data*)

inv_serial_write_mem() - used to write multiple bytes of data to the memory.

a file handle to the serial device used for the communication. I2C slave address of device. The location in the memory to write to. Length of burst data. Pointer to block of data.

returns INV_SUCCESS == 0 if successful; a non-zero error code otherwise.

5.16 MLERROR

Definition of the error codes used within the MPL and returned to the user.

Definition of the error codes used within the MPL and returned to the user.

Every function tries to return a meaningful error code basing on the occuring error condition. The error code is numeric.

The available error codes and their associated values are:

- (0) INV_SUCCESS
- (32) INV_ERROR
- (22 / EINVAL) INV_ERROR_INVALID_PARAMETER
- (1 / EPERM) INV_ERROR_FEATURE_NOT_ENABLED
- (36) INV_ERROR_FEATURE_NOT_IMPLEMENTED
- (38) INV_ERROR_DMP_NOT_STARTED
- (39) INV_ERROR_DMP_STARTED
- (40) INV_ERROR_NOT_OPENED
- (41) INV_ERROR_OPENED
- (19 / ENODEV) INV_ERROR_INVALID_MODULE
- (12 / ENOMEM) INV_ERROR_MEMORY_EXAUSTED
- (44) INV_ERROR_DIVIDE_BY_ZERO
- (45) INV_ERROR_ASSERTION_FAILURE
- (46) INV_ERROR_FILE_OPEN
- (47) INV_ERROR_FILE_READ
- (48) INV_ERROR_FILE_WRITE
- (49) INV_ERROR_INVALID_CONFIGURATION
- (52) INV_ERROR_SERIAL_CLOSED
- (53) INV_ERROR_SERIAL_OPEN_ERROR
- (54) INV_ERROR_SERIAL_READ
- (55) INV_ERROR_SERIAL_WRITE
- (56) INV_ERROR_SERIAL_DEVICE_NOT_RECOGNIZED

- (57) INV_ERROR_SM_TRANSITION
- (58) INV_ERROR_SM_IMPROPER_STATE
- (62) INV_ERROR_FIFO_OVERFLOW
- (63) INV_ERROR_FIFO_FOOTER
- (64) INV_ERROR_FIFO_READ_COUNT
- (65) INV_ERROR_FIFO_READ_DATA
- (72) INV_ERROR_MEMORY_SET
- (82) INV_ERROR_LOG_MEMORY_ERROR
- (83) INV_ERROR_LOG_OUTPUT_ERROR
- (92) INV_ERROR_OS_BAD_PTR
- (93) INV_ERROR_OS_BAD_HANDLE
- (94) INV_ERROR_OS_CREATE_FAILED
- (95) INV_ERROR_OS_LOCK_FAILED
- (102) INV_ERROR_COMPASS_DATA_OVERFLOW
- (103) INV_ERROR_COMPASS_DATA_UNDERFLOW
- (104) INV_ERROR_COMPASS_DATA_NOT_READY
- (105) INV_ERROR_COMPASS_DATA_ERROR
- (107) INV_ERROR_CALIBRATION_LOAD
- (108) INV_ERROR_CALIBRATION_STORE
- (109) INV_ERROR_CALIBRATION_LEN
- (110) INV_ERROR_CALIBRATION_CHECKSUM
- (111) INV_ERROR_ACCEL_DATA_OVERFLOW
- (112) INV_ERROR_ACCEL_DATA_UNDERFLOW
- (113) INV_ERROR_ACCEL_DATA_NOT_READY
- (114) INV_ERROR_ACCEL_DATA_ERROR

The available warning codes and their associated values are:

- (115) INV_WARNING_MOTION_RACE
- (116) INV_WARNING_QUAT_TRASHED

5.17 FAST_NO_MOT

Fast no motion algorithm.

Files

- file fastNoMotion.c

  Fast no motion algorithm.

Functions

- void int_set_fast_nomot_gyro_threshold (float thresh)

  Sets internal threshold for fast no motion.

- inv_error_t inv_disable_fast_nomot (void)

  Turns off the FastNoMotion feature.

- inv_error_t inv_enable_fast_nomot (void)

  Turns on a faster Motion/No Motion.

- inv_error_t inv_fast_nomot_is_enabled (unsigned char *is_enabled)

  inv_fast_nomot_is_enabled *checks if the Fast No Motion algorithm is currently being used to update the gyro biases.*

- void inv_get_fast_nomot_accel_param (long *cntr, float *param)

  This is used to help set inv_set_fast_nomot_accel_threshold().

- void inv_get_fast_nomot_compass_param (long *cntr, float *param)

  This is used to help set inv_set_fast_nomot_compass_threshold().

- void inv_set_fast_nomot_accel_threshold (float thresh)

  Used to set internal threshold.

- void inv_set_fast_nomot_compass_threshold (float thresh)

  Used to set internal threshold.

5.17.1 Detailed Description

Fast no motion algorithm.

5.17.2 Function Documentation

5.17.2.1 inv_error_t inv_enable_fast_nomot (void)

Turns on a faster Motion/No Motion.

Sometimes it has false Motions and is intended to be used to set Gyro Biases quickly. This will turn off

5.17.2.2 inv_error_t inv_fast_nomot_is_enabled (unsigned char * *is_enabled*)

inv_fast_nomot_is_enabled checks if the Fast No Motion algorithm is currently being used to update the gyro biases.

Parameters:

is_enabled True if Fast No Motion is enabled.

Returns:

INV_SUCCESS if successful.

5.17.2.3 void inv_get_fast_nomot_accel_param (long * *cntr*, float * *param*)

This is used to help set inv_set_fast_nomot_accel_threshold().

cntr is incremented each time there is a new value of param. 100 new values should be sorted from low to high and the 97th value should be used as the threshold in inv_set_fast_nomot_accel_threshold(). The compass must be on.

5.17.2.4 void inv_get_fast_nomot_compass_param (long * *cntr*, float * *param*)

This is used to help set inv_set_fast_nomot_compass_threshold().

cntr is incremented each time there is a new value of param. 100 new values should be sorted from low to high and the 97th value should be used as the threshold in inv_set_fast_nomot_compass_threshold(). The compass must be on.

5.17.2.5 void inv_set_fast_nomot_accel_threshold (float *thresh*)

Used to set internal threshold.

This may need to be set based upon device environment. See inv_get_fast_nomot_accel_param() for values a range of values to set this too.

Parameters:

thresh

5.17.2.6 void inv_set_fast_nomot_compass_threshold (float *thresh*)

Used to set internal threshold.

This may need to be set based upon device environment. See inv_get_fast_nomot_compass_param() for values a range of values to set this too.

Parameters:

thresh

5.18 PLUGIN_GESTURE

Motion Library - Gesture plugin.

Classes

- struct tGesture

  Gesture data structure.

Modules

- PLUGIN_TAP

  Motion Library - Gesture Engine - Tap Detection Algorithm.

- PLUGIN_SHAKE

  Motion Library - Gesture Engine - Shake Detection Algorithm.

- PLUGIN_YAW_ROTATE

  Motion Library - Gesture Engine - Yaw Rotation Detection Algorithm.

Files

- file gesture.c

  Gesture Library Implemenation.

5.18.1 Detailed Description

Motion Library - Gesture plugin.

The Gesture Library processes gyroscopes and accelerometers to provide recognition of a set of gestures. These include tapping, shaking along various axes, and rotation about a horizontal axis.

Note:

This feature feature is available as a plugin on top of the regular MPL solution.

5.19 PLUGIN_TAP

Motion Library - Gesture Engine - Tap Detection Algorithm.

Classes

- struct tGestureTap

  Tap gesture data structure.

5.19.1 Detailed Description

Motion Library - Gesture Engine - Tap Detection Algorithm.

Tap allows detection of one or more sequential taps. Call inv_enable_gesture() and then inv_set_gesture() using INV_TAP to enable tap detection.

See also:

GESTURE

5.20 PLUGIN_SHAKE

Motion Library - Gesture Engine - Shake Detection Algorithm.

Classes

- struct tGestureShake

  Shake gesture data structure.

5.20.1 Detailed Description

Motion Library - Gesture Engine - Shake Detection Algorithm.

Shake allows detection of one or more sequential shakes. Call inv_enable_gesture() and then inv_set_gesture() using INV_PITCH_SHAKE, INV_ROLL_SHAKE, INV_-YAW_SHAKE, and/or INV_SHAKE_ALL to enable shake detection.

See also:

GESTURE

5.21 PLUGIN_YAW_ROTATE

Motion Library - Gesture Engine - Yaw Rotation Detection Algorithm.

Classes

- struct tGestureYawImageRotate

  Yaw image rotate gesture data structure.

5.21.1 Detailed Description

Motion Library - Gesture Engine - Yaw Rotation Detection Algorithm.

Yaw allows detection of one ore more sequential yaw roations. Call inv_enable_-gesture() and then inv_set_gesture() using INV_YAW_IMAGE_ROTATE to enable yaw rotation detection.

See also:

GESTURE

5.22 PLUGIN_ORIENTATION

Motion Library - Orientation plugin.

Files

- file orientation.c

  Determines the orientation of the device.

Functions

- inv_error_t inv_disable_orientation (void)

  Turns off the orientation feature.

- inv_error_t inv_enable_orientation (void)

  Turns on the orientation feature.

- inv_error_t inv_get_orientation (int *orientation)

  Gets the last reported orientation.

- inv_error_t inv_get_orientation_state (int *state)

  Used to retrieve the state of the orientation engine.

- inv_error_t inv_set_orientation (int orientation)

  Used to register which orientations will trigger the user defined callback function.

- inv_error_t inv_set_orientation_cb (void(*callback)(unsigned short))

  Sets the callback for the orientation changes.

- inv_error_t inv_set_orientation_interrupt (unsigned char on)

  turns on interrupt generation when there is an orientation change

- inv_error_t inv_set_orientation_thresh (float angle, float hysteresis, unsigned long time, unsigned int axis)

  Sets the threshold for the orientation angle, hysteresis and time.

5.22.1 Detailed Description

Motion Library - Orientation plugin.

Report when the orientation of the device changes.

Note:

This feature feature is available as a plugin on top of the regular MPL solution.

Orientation functions are available after calling inv_dmp_open() with either MLDmpDefaultOpen() or inv_open_low_power_pedometer().

- `inv_dmp_open( MLDmpDefaultOpen );`
- `inv_dmp_open( inv_open_low_power_pedometer );`

These functions allow for changes in orientation to be detected. To use, first open using inv_enable_orientation() then set the list of orientations to be detected using inv_set_orientation(), then set the callback with which to be notified when the orientation changes using inv_set_orientation_cb(). See example below:

```
void OrientationCallback(unsigned short newOrientation)
{
   // Do something with newOrientation
}

// ...

// Set up orientation
result = inv_enable_orientation();
if (INV_SUCCESS != result)
{
    // Handle error condition
}

// Enable detection of all orientations
result = inv_set_orientation(INV_ORIENTATION_ALL);
if (INV_SUCCESS != result)
{
    // Handle error condition
}

result = inv_set_orientation_cb(OrientationCallback);
if (INV_SUCCESS != result)
{
    // Handle error condition
}
```

5.22.2 Function Documentation

5.22.2.1 inv_error_t inv_disable_orientation (void)

Turns off the orientation feature.

Returns:

- INV_SUCCESS if successful
- Non-zero error code on failure.

5.22.2.2 inv_error_t inv_enable_orientation (void)

Turns on the orientation feature.

This will also reset any other orientation function calls. Call before any other orientation function.

Precondition:

inv_dmp_open() Must be called with either MLDmpDefaultOpen() or MLPedometerStandAloneOpen() before calling this function. inv_dmp_start() must **NOT** have been called.

Returns:

- INV_SUCCESS if successful
- Non-zero error code on failure.

5.22.2.3 inv_error_t inv_get_orientation (int * *orientation*)

Gets the last reported orientation.

Can also be used to get the inital orientation in case the device starts in the initial orientation.

Parameters:

orientation One of

- INV_X_UP,
- INV_X_DOWN,
- INV_Y_UP,
- INV_Y_DOWN,
- INV_Z_UP,

- INV_Z_DOWN

Returns:

- INV_SUCCESS if successful
- Non-zero error code on failure.

5.22.2.4 inv_error_t inv_get_orientation_state (int * *state*)

Used to retrieve the state of the orientation engine.

When the orientation engine detects a high probability orientation event, This function will return INV_STATE_RUNNING, indicating that it is actively processing incoming data to determine if a programmed orientation has occured.

Otherwise this fucntion will return INV_STATE_IDLE.

Precondition:

inv_dmp_open() Must be called with MLDmpDefaultOpen() and then inv_dmp_start().

Parameters:

state Sets this value to:

- INV_STATE_IDLE
- INV_STATE_RUNNING if a recent event occured.

Returns:

- INV_SUCCESS correctly called
- Non-zero error code otherwise

5.22.2.5 inv_error_t inv_set_orientation (int *orientation*)

Used to register which orientations will trigger the user defined callback function.

Allows a user to register which orientations will trigger the user defined callback function. Zero is returned if the command is successful; otherwise, an ML error code is returned.

Precondition:

inv_dmp_open() Must be called with either MLDmpDefaultOpen() or MLPedometerStandAloneOpen() before calling this function.

Parameters:

orientation An orientation or bitwise OR of orientations to be detected. The orientations are:

- INV_X_UP,
- INV_X_DOWN,
- INV_Y_UP,
- INV_Y_DOWN,
- INV_Z_UP,
- INV_Z_DOWN, and
- INV_ORIENTATION_ALL: Bitwise or of all previous orientations.

Returns:

INV_SUCCESS if successful, a non-zero error code on failure.

5.22.2.6 inv_error_t inv_set_orientation_cb (void(∗)(unsigned short) *callback*)

Sets the callback for the orientation changes.

Must be called after inv_enable_orientation(). The callback function will have passed as its arguement the new orientation as one of:

- INV_X_UP,
- INV_X_DOWN,
- INV_Y_UP,
- INV_Y_DOWN,
- INV_Z_UP,
- INV_Z_DOWN

Parameters:

callback The callback you want to use for orientation callbacks. Only one callback can be used.

Returns:

- INV_SUCCESS if successful
- Non-zero error code on failure.

5.22.2.7 inv_error_t inv_set_orientation_interrupt (unsigned char *on*)

turns on interrupt generation when there is an orientation change

Parameters:

on Boolean to turn the error code on or off

Returns:

INV_SUCCESS or non-zero error code

5.22.2.8 inv_error_t inv_set_orientation_thresh (float *angle*, float *hysteresis*, unsigned long *time*, unsigned int *axis*)

Sets the threshold for the orientation angle, hysteresis and time.

The angle is defined to be the angle that the gravity vector makes with the normal vector of the body plane in plane of the axis. I.E. if the body plane is (0, 0.5, 0.866) the angles in the respective planes are (0, 30, 60).

Hysteresis is the minimum amount of angle by which the old orientation difference has to exceed the new orientation to cause a change. This value is used by first converting to a vector and then performing the threshold on the gravity vector. Thus the hysteresis value is the minimum angle and is is non-linear with the angle. If y is up, Z is 0, and a hysteresis of 5 degrees (vector sin(5) == 0.087g), x will be up when the gravity vector is (0.749, 0.662, 0) corresponding to (48.5 41.5, 0), with the actual hysteresis being 7 degrees.

The hysteresis is used to determine which axis is up in areas of ambiguity. It is also possible to create an area where no axis is up. This is a dead zone. In this case the orientation will trigger an interrupt if configured and try to determine the axis that is most up. These interrupts and callbacks will continue to fire until one of the axes crosses a threshold.

Parameters:

angle in degrees

hysteresis minimum value in degrees

time time in ms that the device must remain in the new orientation to register an orientation change

axis Which axis to set this threshold for

- INV_X_AXIS
- INV_Y_AXIS
- INV_Z_AXIS

Returns:

INV_SUCCESS or non-zero error code

5.23 PLUGIN_GLYPH

Motion Library - Characters Recognition plugin.

Classes

- struct tMLGlyphData

  Describes the data to be used by the character recognition algorithm.

Files

- file mlglyph.c

  The Control Library.

Functions

- inv_error_t inv_add_glyph (unsigned short glyphID)

  Adds a new glyph.

- inv_error_t inv_best_glyph (unsigned short *finalGlyph)

  Finds the best match for a glyph.

- inv_error_t inv_clear_glyph (void)

  Clears the glyph.

- inv_error_t inv_enable_glyph (void)

  Enables the glyph engine.

- inv_error_t inv_get_glyph (int index, int *x, int *y)

  Returns a trajectory data point.

- inv_error_t inv_get_glyph_length (unsigned short *length)

  Returns the glyph length.

- inv_error_t inv_get_glyph_library_length (unsigned short *length)

  Returns the length of the library of glyphs.

- inv_error_t inv_load_glyphs (unsigned char *libraryData)

  Loads a library of glyphs.

- inv_error_t inv_reset_glyph_library ()

  Resets glyph library.

- inv_error_t inv_set_glyph_prob_thresh (unsigned short prob)

  Sets the minimum recognition probability.

- inv_error_t inv_set_glyph_speed_thresh (unsigned short speed)

  Sets the glyph speed threshold.

- inv_error_t inv_start_glyph (void)

  Turns on motion tracking.

- inv_error_t inv_stop_glyph (void)

  Turns off motion tracking.

- inv_error_t inv_store_glyphs (unsigned char *libraryData, unsigned short *length)

  Stores a library of glyphs.

- inv_error_t MLDisableGlyph (void)

  Disables the glyph engine.

5.23.1 Detailed Description

Motion Library - Characters Recognition plugin.

The glyph library process a series of user trajectories and stores them for later reconition or compares them against previously stored glyphs.

Note:

This feature feature is available as a plugin on top of the regular MPL solution.

The library has two main purpose: training the glyph patterns, and recognizing the stored pattens.

To run the functions in ML_GLYPH library, user first has to call inv_enable_glyph(). Then depends on the mode the code could look something like:

```
[...]
inv_enable_glyph();
[...]
if (TRAINING_MODE) {
```

```
    inv_start_glyph();
    [...]
    inv_add_glyph(GlyphID);
    inv_clear_glyph();
    inv_stop_glyph();
    inv_store_glyphs(LIBRARY_DATA);
} else {
    inv_load_glyphs(LIBRARY_DATA);
    [...]
    inv_start_glyph();
    [...]
    inv_best_glyph(RECOGNIZED_CHARACTER);
    inv_clear_glyph();
    inv_stop_glyph();
}
[...]
```

5.23.2 Function Documentation

5.23.2.1 inv_error_t inv_add_glyph (unsigned short *glyphID*)

Adds a new glyph.

inv_add_glyph extends the library by adding a newly created glyph.

Precondition:

inv_dmp_open() Must be called with MLDmpDefaultOpen() or inv_open_low_power_pedometer()

Parameters:

glyphID The glyph ID to be associated with this glyph.

Returns:

Zero is returned if the command is successful; otherwise, an error code is returned.

5.23.2.2 inv_error_t inv_best_glyph (unsigned short * *finalGlyph*)

Finds the best match for a glyph.

Processes all glyphs stored in the library and returns the best match for the user's trajectory.

If a best match cannot be found or the minimum probability for the trajectory was not met (if a minimum probability threshold for the glyph was provided via inv_set_glyph_prob_thresh()), finalGlyph will point to an invalid glyph instance and the function signal it by returning a non-zero error code.

Precondition:

inv_dmp_open() Must be called with MLDmpDefaultOpen() or inv_open_low_power_pedometer()

Parameters:

finalGlyph The glyph ID associated with the matched glyph.

Returns:

INV_SUCCESS is returned if the command is successful; otherwise, a non-zero error code is returned.

5.23.2.3 inv_error_t inv_clear_glyph (void)

Clears the glyph.

inv_clear_glyph resets the glyph trajectory.

Precondition:

inv_dmp_open() Must be called with MLDmpDefaultOpen() or inv_open_low_power_pedometer()

Returns:

Zero is returned if the command is successful; otherwise, an error code is returned.

5.23.2.4 inv_error_t inv_enable_glyph (void)

Enables the glyph engine.

inv_enable_glyph enables the glyph recognition engine.

Precondition:

inv_dmp_open() Must be called with MLDmpDefaultOpen() or inv_open_low_power_pedometer(). inv_dmp_start() must **NOT** have been called yet.

Returns:

Zero is returned if the command is successful; otherwise, an error code is returned.

5.23.2.5 inv_error_t inv_get_glyph (int *index*, int * *x*, int * *y*)

Returns a trajectory data point.

inv_get_glyph returns the value of the glyph trajectory at a given data point. It can be used for displaying the trajectory in real time for user feedback.

Precondition:

inv_dmp_open() Must be called with MLDmpDefaultOpen() or inv_open_low_-power_pedometer()

Parameters:

index The index of the trajectory data point.

x The x value of the trajectory data point.

y The y value of the trajectory data point.

Returns:

Zero is returned if the command is successful; otherwise, an error code is returned.

5.23.2.6 inv_error_t inv_get_glyph_length (unsigned short * *length*)

Returns the glyph length.

inv_get_glyph_length returns the number of data points in the glyph trajectory.

Precondition:

inv_dmp_open() Must be called with MLDmpDefaultOpen() or inv_open_low_-power_pedometer()

Parameters:

length The length of the glyph trajectory.

Returns:

Zero is returned if the command is successful; otherwise, an error code is returned.

5.23.2.7 inv_error_t inv_get_glyph_library_length (unsigned short * *length*)

Returns the length of the library of glyphs.

Should be called before allocating the memory required to store this data to a file.

Precondition:

inv_dmp_open() Must be called with MLDmpDefaultOpen() or inv_open_low_power_pedometer()

Parameters:

length The length of the library.

Returns:

Zero is returned if the command is successful; otherwise, an error code is returned.

5.23.2.8 inv_error_t inv_load_glyphs (unsigned char * *libraryData*)

Loads a library of glyphs.

MLLoadGlyph parses a binary data set containing a library of glyphs. The binary data set is intended to be loaded from a file.

Precondition:

inv_dmp_open() Must be called with MLDmpDefaultOpen() or inv_open_low_power_pedometer()

Parameters:

libraryData A pointer to an array of bytes to be parsed.

Returns:

Zero is returned if the command is successful; otherwise, an error code is returned.

5.23.2.9 inv_error_t inv_reset_glyph_library ()

Resets glyph library.

inv_reset_glyph_library clears the glyph library of all data.

Precondition:

inv_dmp_open() Must be called with MLDmpDefaultOpen() or inv_open_low_power_pedometer()

Returns:

Zero is returned if the command is successful; otherwise, an error code is returned.

5.23.2.10 inv_error_t inv_set_glyph_prob_thresh (unsigned short *prob*)

Sets the minimum recognition probability.

inv_set_glyph_prob_thresh sets the minimum probability required for returning a successful glyph recognition.

Precondition:

inv_dmp_open() Must be called with MLDmpDefaultOpen() or inv_open_low_power_pedometer()

Parameters:

prob The minimum recognition probability

Returns:

Zero is returned if the command is successful; otherwise, an error code is returned.

5.23.2.11 inv_error_t inv_set_glyph_speed_thresh (unsigned short *speed*)

Sets the glyph speed threshold.

inv_set_glyph_speed_thresh determines the minimum speed at which the user must move in order to influence the glyph trajectory.

Precondition:

inv_dmp_open() Must be called with MLDmpDefaultOpen() or inv_open_low_power_pedometer()

Parameters:

speed The minimum speed threshold

Returns:

Zero is returned if the command is successful; otherwise, an error code is returned.

5.23.2.12 inv_error_t inv_start_glyph (void)

Turns on motion tracking.

inv_start_glyph enables tracking of the user's trajectory, and is intended to be associated with a button press.

Precondition:

inv_dmp_open() Must be called with MLDmpDefaultOpen() or inv_open_low_power_pedometer()

Returns:

Zero is returned if the command is successful; otherwise, an error code is returned.

5.23.2.13 inv_error_t inv_stop_glyph (void)

Turns off motion tracking.

inv_stop_glyph disables tracking of the user's trajectory, and is intended to be associated with a button release.

Precondition:

inv_dmp_open() Must be called with MLDmpDefaultOpen() or inv_open_low_power_pedometer()

Returns:

Zero is returned if the command is successful; otherwise, an error code is returned.

5.23.2.14 inv_error_t inv_store_glyphs (unsigned char * *libraryData*, unsigned short * *length*)

Stores a library of glyphs.

MLStoreGlyph generates a binary data set containing a library of glyphs. The binary data set is intended to be stored to a file.

Precondition:

inv_dmp_open() Must be called with MLDmpDefaultOpen() or inv_open_low_power_pedometer()

Parameters:

libraryData A pointer to an array of bytes to be stored.

length The length of the array of bytes.

Returns:

Zero is returned if the command is successful; otherwise, an error code is returned.

5.23.2.15 inv_error_t MLDisableGlyph (void)

Disables the glyph engine.

MLDisableGlyph disables the glyph recognition engine.

Precondition:

inv_dmp_open() Must be called with MLDmpDefaultOpen() or inv_open_low_-power_pedometer(). inv_dmp_start() must **NOT** have been called yet.

Returns:

Zero is returned if the command is successful; otherwise, an error code is returned.

5.24 PLUGIN_PEDOMETER_STAND_ALONE

Motion Library - Pedometer Stand-Alone plugin.

Files

- file mlpedometer_fullpower.c
  Motion Library - Full Power Pedometer Engine.
- file mlpedometer_lowpower.c
  Motion Library - Low Power Pedometer Engine.
- file pedometerStandAlone.c
  Pedometer stand alone source file.

Functions

- inv_error_t inf_set_full_power_pedometer_step_buffer_reset_time (unsigned int timeMs)
  Set the maximum time to wait for a next step to increment the buffer.
- inv_error_t inv_clear_low_power_pedometer_calories ()
  Used to clear the pedometer's calorie counter.
- inv_error_t inv_close_low_power_pedometer (void)
  Closes the Pedometer engine.
- inv_error_t inv_disable_full_power_pedometer (void)
  Disable the pedometer engine.
- inv_error_t inv_enable_full_power_pedometer (void)
  Registers the Full power pedometer to be initialized just before starting.
- inv_error_t inv_get_full_power_pedometer_step_count (unsigned long *steps)
  Get the current step count.
- inv_error_t inv_get_full_power_pedometer_walk_time (double *timeMs)
  Get the current walk time.
- inv_error_t inv_get_low_power_pedometer_calories (unsigned short *calories)

Used to calculate the number of calories burned by the user.

- inv_error_t inv_get_low_power_pedometer_num_of_steps (unsigned long *steps)

  Used retrieve the number of steps taken by the user.

- inv_error_t inv_get_low_power_pedometer_walk_time (double *time)

  Used retrieve the number of steps taken by the user.

- inv_error_t inv_open_low_power_pedometer (void)

  Open up the Stand Alone Pedometer.

- inv_error_t inv_set_full_power_pedometer_params (const struct stepParams *params)

  Set the parameters to use for the full power pedometer.

- inv_error_t inv_set_full_power_pedometer_step_buffer (unsigned short minSteps)

  Set the number of steps to buffer before reporting new steps.

- inv_error_t inv_set_full_power_pedometer_step_callback (void(*func)(unsigned long stepNum, double walkTimeMs))

  Set a function to be called every time a step is detected.

- inv_error_t inv_set_full_power_pedometer_step_count (unsigned long steps)

  Set the number of steps taken so far.

- inv_error_t inv_set_full_power_pedometer_walk_time (double timeMs)

  Set the amount of time walked so far.

- inv_error_t inv_set_low_power_pedometer_buffer_reset_time (unsigned int timeMs)

  Set the maximum time to wait for a next step to increment the buffer.

- inv_error_t inv_set_low_power_pedometer_num_of_steps (unsigned long steps)

  Used to set the initial step count.

- inv_error_t inv_set_low_power_pedometer_step_buffer (unsigned short minSteps)

  Set the number of steps to buffer before reporting new steps.

- inv_error_t inv_set_low_power_pedometer_stride_length (unsigned short stride-Length)

  Used to set the user's stride length.

- inv_error_t inv_set_low_power_pedometer_walk_time (double time)

  Used to set the initial step count.

- inv_error_t inv_set_low_power_pedometer_weight (unsigned short weight)

  Used to set the user's weight.

- inv_error_t inv_start_low_power_pedometer (void)

  Start the DMP.

- inv_error_t inv_stop_low_power_pedometer (void)

  Stops the DMP and puts it in low power.

- inv_error_t MLPedometerSetNoMotionThresh (float thresh)

  inv_set_no_motion_thresh is used to set the threshold for detecting INV_NO_-MOTION

- inv_error_t MLPedometerSetNoMotionTime (float time)

  inv_set_no_motion_time is used to set the time required for detecting INV_NO_-MOTION

5.24.1 Detailed Description

Motion Library - Pedometer Stand-Alone plugin.

Motion Library - Low Power Pedometer plugin.

Motion Library - Full Power Pedometer plugin.

The Pedometer application counts steps by the user. The pedometer application can not be used with most of the gesture and tap features or the cross axis support. The use of gyros are optional and may be turned off for power savings.

Note:

This feature feature is available as a plugin on top of the regular MPL solution.

The Pedometer full power engine counts steps by the user using the computation power from the application processor. The MPU's DMP processor is still used to execute some complex computation but the use of the host processor will allow to use some augmented functionalities such as the gesture engine while still countin the steps.

Note:

This feature feature is available as a plugin on top of the regular MPL solution.

The Pedometer low power engine counts steps by the user offloading the computation power from the application processor to the MPU's processor. For this reason the feature supported in this mode are limited to the step counting.

Note:

This feature feature is available as a plugin on top of the regular MPL solution.

5.24.2 Function Documentation

5.24.2.1 inv_error_t inf_set_full_power_pedometer_step_buffer_reset_time (unsigned int *timeMs*)

Set the maximum time to wait for a next step to increment the buffer.

Parameters:

timeMs How lon in ms

Returns:

INV_SUCCESS or non-zero error code

5.24.2.2 inv_error_t inv_clear_low_power_pedometer_calories ()

Used to clear the pedometer's calorie counter.

Clears the counter

Precondition:

inv_dmp_open() with MLDmpPedometerStandAlone() must have been called.

Returns:

INV_SUCCESS or non-zero error code

5.24.2.3 inv_error_t inv_close_low_power_pedometer (void)

Closes the Pedometer engine.

Does not close the serial communication. To do that, call inv_serial_stop(). After calling inv_close_low_power_pedometer() another DMP module can be loaded in the MPL with the corresponding necessary intialization and configurations, via any of the inv_dmp_openXXX functions.

Precondition:

inv_open_low_power_pedometer() must have been called.

Returns:

INV_SUCCESS, Non-zero error code otherwise.

5.24.2.4 inv_error_t inv_disable_full_power_pedometer (void)

Disable the pedometer engine.

Returns:

INV_SUCCESS or non-zero error code

5.24.2.5 inv_error_t inv_enable_full_power_pedometer (void)

Registers the Full power pedometer to be initialized just before starting.

Note:

inv_dmp_start will return INV_ERROR_INVALID_CONFIGURATION if inv_set_fifo_rate was called with a value 10 or greater

The full power pedometer needs to know the data rate, so initialization is delayed until the inv_dmp_start is called at which time initialization is performed.

Precondition:

inv_set_fifo_rate must be set to a value [0..9] before inv_dmp_start is called Must be called before inv_dmp_start

Returns:

ML_SUCCES or non-zero error code

5.24.2.6 inv_error_t inv_get_full_power_pedometer_step_count (unsigned long * *steps*)

Get the current step count.

Parameters:

steps Curren step count

Returns:

INV_SUCCESS or non-zero error code

5.24.2.7 inv_error_t inv_get_full_power_pedometer_walk_time (double * *timeMs*)

Get the current walk time.

Parameters:

timeMs current time walking

Returns:

INV_SUCCESS or non-zero error code

5.24.2.8 inv_error_t inv_get_low_power_pedometer_calories (unsigned short * *calories*)

Used to calculate the number of calories burned by the user.

It depends on the number of steps taken, the weight of the user, and the stride length of the user.

Precondition:

inv_dmp_open() with MLDmpPedometerStandAlone() must have been called.

Parameters:

calories number of calories buffer

Returns:

INV_SUCCESS or non-zero error code

5.24.2.9 inv_error_t inv_get_low_power_pedometer_num_of_steps (unsigned long * *steps*)

Used retrieve the number of steps taken by the user.

Typically it is used for polling mode.

Precondition:

inv_dmp_open() with MLDmpPedometerStandAlone() must have been called.

Parameters:

steps The number of steps taken by the user.

Returns:

INV_SUCCESS or non-zero error code

5.24.2.10 inv_error_t inv_get_low_power_pedometer_walk_time (double * *time*)

Used retrieve the number of steps taken by the user.

Typically it is used for polling mode.

Precondition:

inv_dmp_open() with MLDmpPedometerStandAlone() must have been called.

Parameters:

steps The number of steps taken by the user.

Returns:

INV_SUCCESS or non-zero error code

5.24.2.11 inv_error_t inv_open_low_power_pedometer (void)

Open up the Stand Alone Pedometer.

You may only have one motion sensor engine open. This function is mutually exclusive with inv_dmp_open(). inv_serial_start() should be called before calling this to open up the communication layer.

Returns:

INV_SUCCESS or non-zero error code

5.24.2.12 inv_error_t inv_set_full_power_pedometer_params (const struct stepParams * *params*)

Set the parameters to use for the full power pedometer.

Parameters:

params the parameters

Returns:

INV_SUCCESS or non-zero error code.

5.24.2.13 inv_error_t inv_set_full_power_pedometer_step_buffer (unsigned short *minSteps*)

Set the number of steps to buffer before reporting new steps.

NOTE: This is the number of steps to buffer BEFORE reporting new steps. Thus if the value is set to 5, the 6th step will be reported

Parameters:

minSteps The number of steps to buffer before reporing new steps

Returns:

INV_SUCCESS or non-zero error code

5.24.2.14 inv_error_t inv_set_full_power_pedometer_step_callback (void(*)(unsigned long stepNum, double walkTimeMs) *func*)

Set a function to be called every time a step is detected.

Parameters:

func a pointer to a function taking an unsigned long to be called

Returns:

INV_SUCCESS or non-zero error code.

5.24.2.15 inv_error_t inv_set_full_power_pedometer_step_count (unsigned long *steps*)

Set the number of steps taken so far.

Parameters:

steps

Returns:

5.24.2.16 inv_error_t inv_set_full_power_pedometer_walk_time (double *timeMs*)

Set the amount of time walked so far.

Parameters:

timeMs time walked in ms

Returns:

INV_SUCCESS or non-zero error code

5.24.2.17 inv_error_t inv_set_low_power_pedometer_buffer_reset_time (unsigned int *timeMs*)

Set the maximum time to wait for a next step to increment the buffer.

Parameters:

timeMs How lon in ms

Returns:

INV_SUCCESS or non-zero error code

5.24.2.18 inv_error_t inv_set_low_power_pedometer_num_of_steps (unsigned long *steps*)

Used to set the initial step count.

If the pedometer has been off for a while, use this to set the initial step count.

Parameters:

steps The number of steps to start from.

Returns:

INV_SUCCESS or non-zero error code otherwise.

5.24.2.19 inv_error_t inv_set_low_power_pedometer_step_buffer (unsigned short *minSteps*)

Set the number of steps to buffer before reporting new steps.

NOTE: This is the number of steps to buffer BEFORE reporting new steps. Thus if the value is set to 5, the 6th step will be reported

Parameters:

minSteps The number of steps to buffer before reporing new steps

Returns:

INV_SUCCESS or non-zero error code

5.24.2.20 inv_error_t inv_set_low_power_pedometer_stride_length (unsigned short *strideLength*)

Used to set the user's stride length.

Value will be used in the calorie calculation.

Precondition:

inv_dmp_open() with MLDmpPedometerStandAlone() must have been called.

Parameters:

strideLength User stride length in centimeters

Returns:

INV_SUCCESS or non-zero error code

5.24.2.21 inv_error_t inv_set_low_power_pedometer_walk_time (double *time*)

Used to set the initial step count.

If the pedometer has been off for a while, use this to set the initial step count.

Parameters:

steps The number of steps to start from.

Returns:

INV_SUCCESS or non-zero error code otherwise.

5.24.2.22 inv_error_t inv_set_low_power_pedometer_weight (unsigned short *weight*)

Used to set the user's weight.

Value will be used in the calorie calculation.

Precondition:

inv_dmp_open() with MLDmpPedometerStandAlone() must have been called.

Parameters:

weight User's weight to use in kilograms

Returns:

INV_SUCCESS or non-zero error code

5.24.2.23 inv_error_t inv_start_low_power_pedometer (void)

Start the DMP.

Precondition:

inv_dmp_open() must have been called.

Returns:

INV_SUCCESS if successful, or Non-zero error code otherwise.

5.24.2.24 inv_error_t inv_stop_low_power_pedometer (void)

Stops the DMP and puts it in low power.

Precondition:

inv_dmp_start() must have been called.

Returns:

INV_SUCCESS, Non-zero error code otherwise.

5.24.2.25 inv_error_t MLPedometerSetNoMotionThresh (float *thresh*)

inv_set_no_motion_thresh is used to set the threshold for detecting INV_NO_-MOTION

Precondition:

inv_dmp_open() or inv_open_low_power_pedometer() and inv_dmp_start() must **NOT** have been called.

Parameters:

thresh A threshold scaled in g

Returns:

INV_SUCCESS if successful or Non-zero error code otherwise.

5.24.2.26 inv_error_t MLPedometerSetNoMotionTime (float *time*)

inv_set_no_motion_time is used to set the time required for detecting INV_NO_-MOTION

Precondition:

inv_dmp_open() or inv_open_low_power_pedometer() and inv_dmp_start() must **NOT** have been called.

Parameters:

time A time in seconds.

Returns:

INV_SUCCESS if successful or Non-zero error code otherwise.

5.25 NINEAXIS_SENSOR_FUSION

Advanced sensor fusion functionalities, including 9 axis compass sensor fusion and temperature bias compensation.

Files

- file mlsupervisor_9axis.c

  Advanced sensor fusion functionalities, including 9 axis compass sensor fusion and temperature bias compensation.

Functions

- void examine_large_mag_field (void)

  Simple check if we are in a large magnetic field and sets the inv_obj.adv_fusion->large_field apropriately.

- inv_error_t inv_disable_9x_fusion (void)

  Disable the compass interaction with sensor fusion and all other advanced proprietary motion processing features.

- inv_error_t inv_disable_9x_fusion_basic (void)

  Disable the compass interaction with sensor fusion and all other advanced proprietary motion processing features.

- inv_error_t inv_disable_9x_fusion_external (void)

  Disable the compass interaction with sensor fusion when a 3rd party compass calibration library is in use.

- inv_error_t inv_disable_9x_fusion_legacy (void)

  Disable the compass interaction with sensor fusion and all other advanced proprietary motion processing features.

- inv_error_t inv_disable_9x_fusion_new (void)

  Disables the compass interaction with sensor fusion and all other advanced proprietary motion processing features.

- inv_error_t inv_disable_maintain_heading (void)

  Disable only the tracking of heading using the compass correction available.

- inv_error_t inv_enable_9x_fusion (void)

Enable the compass interaction with sensor fusion and all other advanced proprietary motion processing features.

- inv_error_t inv_enable_9x_fusion_basic (void)

  Enable the compass interaction with sensor fusion and all other advanced proprietary motion processing features.

- inv_error_t inv_enable_9x_fusion_external (void)

  Enable the compass interaction with sensor fusion and keep the proprietary compass calibration disable.

- inv_error_t inv_enable_9x_fusion_legacy (void)

  Enable the compass interaction with sensor fusion and all other advanced proprietary motion processing features.

- inv_error_t inv_enable_9x_fusion_new (void)

  Enable the compass interaction with sensor fusion and all other advanced proprietary motion processing features.

- inv_error_t inv_enable_maintain_heading (void)

  Enable only the tracking of heading only using the compass correction available.

5.25.1 Detailed Description

Advanced sensor fusion functionalities, including 9 axis compass sensor fusion and temperature bias compensation.

5.25.2 Function Documentation

5.25.2.1 inv_error_t inv_disable_9x_fusion (void)

Disable the compass interaction with sensor fusion and all other advanced proprietary motion processing features.

Note:

Turns off 9axis sensor fusion, magnetic disturbance detection and compass bias determination. The compass bias algorithm is picked by the type of compass that is detected.
If a compass is not setup in the system or is not detected, some funtionalities will not be available.

Returns:

INV_SUCCESS if the advanced sensor fusion functionalities were successfully disabled. INV_ERROR_INVALID_MODULE or other non-zero error code otherwise.

5.25.2.2 inv_error_t inv_disable_9x_fusion_basic (void)

Disable the compass interaction with sensor fusion and all other advanced proprietary motion processing features.

Note:

Turns off 9axis sensor fusion, magnetic disturbance detection and compass bias determination.
If a compass is not setup in the system or is not detected, some funtionalities will not be available.

Returns:

INV_SUCCESS if the advanced sensor fusion functionalities were successfully disabled. INV_ERROR_INVALID_MODULE or other non-zero error code otherwise.

5.25.2.3 inv_error_t inv_disable_9x_fusion_external (void)

Disable the compass interaction with sensor fusion when a 3rd party compass calibration library is in use.

Note:

Turns on 9axis sensor fusion.
If a compass is not setup in the system or is not detected, some funtionalities will not be available.

Returns:

INV_SUCCESS if the advanced sensor fusion functionalities were successfully enabled. INV_ERROR_INVALID_MODULE or other non-zero error code otherwise.

5.25.2.4 inv_error_t inv_disable_9x_fusion_legacy (void)

Disable the compass interaction with sensor fusion and all other advanced proprietary motion processing features.

Note:

Turns off 9axis sensor fusion, magnetic disturbance detection and compass bias determination. The compass bias algorithm has been around for many releases.
If a compass is not setup in the system or is not detected, some funtionalities will not be available.

Returns:

INV_SUCCESS if the advanced sensor fusion functionalities were successfully disabled. INV_ERROR_INVALID_MODULE or other non-zero error code otherwise.

5.25.2.5 inv_error_t inv_disable_9x_fusion_new (void)

Disables the compass interaction with sensor fusion and all other advanced proprietary motion processing features.

Note:

Turns off 9axis sensor fusion, magnetic disturbance detection and compass bias determination. The compass bias algorithm is new for 4.1 and may require some user threshold setting for some compasses.
If a compass is not setup in the system or is not detected, some funtionalities will not be available.

Returns:

INV_SUCCESS if the advanced sensor fusion functionalities were successfully disabled. INV_ERROR_INVALID_MODULE or other non-zero error code otherwise.

5.25.2.6 inv_error_t inv_disable_maintain_heading (void)

Disable only the tracking of heading using the compass correction available.

If enabled, this feature should be disabled prior to enabling any type of 9 axis sensor fusion. When enabling 9 axis sensor fusion maintain_heading is enabled as part of it.

Note:

If a compass is not setup in the system or is not detected, some funtionalities will not be available.

Returns:

INV_SUCCESS if the advanced sensor fusion functionalities were successfully enabled. INV_ERROR_INVALID_MODULE or other non-zero error code otherwise.

5.25.2.7 inv_error_t inv_enable_9x_fusion (void)

Enable the compass interaction with sensor fusion and all other advanced proprietary motion processing features.

Note:

Turns on 9axis sensor fusion, magnetic disturbance detection and compass bias determination. The compass bias algorithm is picked by the type of compass that is detected.
If a compass is not setup in the system or is not detected, some funtionalities will not be available.

Returns:

INV_SUCCESS if the advanced sensor fusion functionalities were successfully enabled. INV_ERROR_INVALID_MODULE or other non-zero error code otherwise.

5.25.2.8 inv_error_t inv_enable_9x_fusion_basic (void)

Enable the compass interaction with sensor fusion and all other advanced proprietary motion processing features.

Note:

Turns on 9axis sensor fusion, magnetic disturbance detection and compass bias determination. The main compass bias determination is not enabled but may be with a call to either: inv_enable_vector_compass_cal() or to inv_enable_compass_fit().
If a compass is not setup in the system or is not detected, some funtionalities will not be available.

Returns:

INV_SUCCESS if the advanced sensor fusion functionalities were successfully enabled. INV_ERROR_INVALID_MODULE or other non-zero error code otherwise.

5.25.2.9 inv_error_t inv_enable_9x_fusion_external (void)

Enable the compass interaction with sensor fusion and keep the proprietary compass calibration disable.

Intended to be used when a 3rd party compass calibration library is integrated within MPL.

Note:

Turns on 9axis sensor fusion only.
If a compass is not setup in the system or is not detected, some funtionalities will not be available.

Returns:

INV_SUCCESS if the advanced sensor fusion functionalities were successfully enabled. INV_ERROR_INVALID_MODULE or other non-zero error code otherwise.

5.25.2.10 inv_error_t inv_enable_9x_fusion_legacy (void)

Enable the compass interaction with sensor fusion and all other advanced proprietary motion processing features.

Note:

Turns on 9axis sensor fusion, magnetic disturbance detection and compass bias determination. The compass bias algorithm has been around for many releases.
If a compass is not setup in the system or is not detected, some funtionalities will not be available.

Returns:

INV_SUCCESS if the advanced sensor fusion functionalities were successfully enabled. INV_ERROR_INVALID_MODULE or other non-zero error code otherwise.

5.25.2.11 inv_error_t inv_enable_9x_fusion_new (void)

Enable the compass interaction with sensor fusion and all other advanced proprietary motion processing features.

Note:

Turns on 9axis sensor fusion, magnetic disturbance detection and compass bias determination. The compass bias algorithm is new for 4.1 and may require some user threshold setting for some compasses.
If a compass is not setup in the system or is not detected, some funtionalities will not be available.

Returns:

INV_SUCCESS if the advanced sensor fusion functionalities were successfully enabled. INV_ERROR_INVALID_MODULE or other non-zero error code otherwise.

5.25.2.12 inv_error_t inv_enable_maintain_heading (void)

Enable only the tracking of heading only using the compass correction available.

This feature should be enabled when no type of 9 axis sensor fusion is already enabled, as it is already enabled as part of those: when on, maintain_heading will movify the quaternion using the compass correction available.

Note:

If a compass is not setup in the system or is not detected, some funtionalities will not be available.

Returns:

INV_SUCCESS if the advanced sensor fusion functionalities were successfully enabled. INV_ERROR_INVALID_MODULE or other non-zero error code otherwise.

Chapter 6

Class Documentation

6.1 ami_chipinfo Struct Reference

AMI chip information ex) 1)model 2)s/n 3)ver 4)more info in the chip.

`#include <ami_sensor_def.h>`

6.1.1 Detailed Description

AMI chip information ex) 1)model 2)s/n 3)ver 4)more info in the chip.

6.2 ami_driverinfo Struct Reference

AMI Driver Information.

`#include <ami_sensor_def.h>`

6.2.1 Detailed Description

AMI Driver Information.

6.3 ami_interference Struct Reference

axis interference information

`#include <ami_sensor_def.h>`

Public Attributes

- signed short xy
  < Y-axis magnetic field for X-axis correction value
- signed short xz
  X-axis magnetic field for Y-axis correction value.
- signed short yx
  Z-axis magnetic field for Y-axis correction value.
- signed short yz
  X-axis magnetic field for Z-axis correction value.
- signed short zx
  Y-axis magnetic field for Z-axis correction value.

6.3.1 Detailed Description

axis interference information

6.3.2 Member Data Documentation

6.3.2.1 signed short ami_interference::xy

< Y-axis magnetic field for X-axis correction value

Z-axis magnetic field for X-axis correction value

6.4 ami_sensor_parametor Struct Reference

sensor calibration Parameter information

`#include <ami_sensor_def.h>`

Public Attributes

- struct ami_vector3d m_gain
  < *geomagnetic field sensor gain*
- struct ami_vector3d m_gain_cor
  geomagnetic field sensor offset
- struct ami_vector3d m_offset
  geomagnetic field sensor axis interference parameter

6.4.1 Detailed Description

sensor calibration Parameter information

6.4.2 Member Data Documentation

6.4.2.1 struct ami_vector3d ami_sensor_parametor::m_gain [read]

< geomagnetic field sensor gain

geomagnetic field sensor gain correction parameter

6.5 ami_sensor_rawvalue Struct Reference

G2-Sensor measurement value (voltage ADC value).

`#include <ami_sensor_def.h>`

Public Attributes

- unsigned short mx

  < geomagnetic field sensor measurement X-axis value (mounted position/direction reference)

- unsigned short my

  geomagnetic field sensor measurement Z-axis value (mounted position/direction reference)

- unsigned short mz

  temperature sensor measurement value

6.5.1 Detailed Description

G2-Sensor measurement value (voltage ADC value).

6.5.2 Member Data Documentation

6.5.2.1 unsigned short ami_sensor_rawvalue::mx

< geomagnetic field sensor measurement X-axis value (mounted position/direction reference)

geomagnetic field sensor measurement Y-axis value (mounted position/direction reference)

6.6 ami_vector3d Struct Reference

axis sensitivity(gain) calibration parameter information

`#include <ami_sensor_def.h>`

Public Attributes

- signed short x
  X-axis.
- signed short y
  Y-axis.
- signed short z
  Z-axis.

6.6.1 Detailed Description

axis sensitivity(gain) calibration parameter information

6.7 ami_win_parameter Struct Reference

Window function Parameter information.

`#include <ami_sensor_def.h>`

Public Attributes

- struct ami_vector3d m_fine
  < *current fine value*
- struct ami_vector3d m_fine_output
  fine value at zero gauss

6.7.1 Detailed Description

Window function Parameter information.

6.7.2 Member Data Documentation

6.7.2.1 struct ami_vector3d ami_win_parameter::m_fine [read]

< current fine value

change per 1coarse

6.8 ext_slave_descr Struct Reference

Description of the slave device for programming.

`#include <mpu.h>`

6.8.1 Detailed Description

Description of the slave device for programming.

Defines the functions and information about the slave the mpu3050 and mpu6050 needs to use the slave device.

Parameters:

init function used to preallocate memory used by the driver.

exit function used to free memory allocated for the driver.

suspend function pointer to put the device in suspended state.

resume function pointer to put the device in running state.

read function that reads the device data.

config function used to configure the device.

get_config function used to get the device's configuration.

name text name of the device.

type device type. enum ext_slave_type

id enum ext_slave_id.

read_reg starting register address to retrieve data.

read_len length in bytes of the sensor data. Typically 6.

endian byte order of the data. enum ext_slave_endian

range full scale range of the slave ouput: struct fix_pnt_range.

trigger If reading data first requires writing a register this is the data to write.

```
struct ext_slave_descr {
  int (* init) (void *mlsl_handle,struct ext_slave_descr *slave,struct
  ext_slave_platform_data *pdata);
  int (* exit) (void *mlsl_handle,struct ext_slave_descr *slave,struct
  ext_slave_platform_data *pdata);
  int (* suspend) (void *mlsl_handle,struct ext_slave_descr *slave,struct
  ext_slave_platform_data *pdata);
  int (* resume) (void *mlsl_handle,struct ext_slave_descr *slave,struct
  ext_slave_platform_data *pdata);
  int (* read) (void *mlsl_handle,struct ext_slave_descr *slave,struct
  ext_slave_platform_data *pdata,__u8 *data);
  int (* config) (void *mlsl_handle,struct ext_slave_descr *slave,struct
  ext_slave_platform_data *pdata,struct ext_slave_config *config);
```

```
  int (* get_config) (void *mlsl_handle,struct ext_slave_descr *slave,struct
  ext_slave_platform_data *pdata,struct ext_slave_config *config);
  char * name;
  __u8 type;
  __u8 id;
  __u8 read_reg;
  __u8 read_len;
  __u8 endian;
  struct fix_pnt_range range;
  struct ext_slave_read_trigger * trigger;
};
```

6.9 ext_slave_platform_data Struct Reference

Platform data for mpu3050 and mpu6050 slave devices.

`#include <mpu.h>`

6.9.1 Detailed Description

Platform data for mpu3050 and mpu6050 slave devices.

The orientation matricies are 3x3 rotation matricies that are applied to the data to rotate from the mounting orientation to the platform orientation. The values must be one of 0, 1, or -1 and each row and column should have exactly 1 non-zero value.

Parameters:

type the type of slave device based on the enum ext_slave_type definitions.

irq the irq number attached to the slave if any.

adapt_num the I2C adapter number.

bus the bus the slave is attached to: enum ext_slave_bus.

address the I2C slave address of the slave device.

orientation[9] the mounting matrix of the device relative to MPU.

irq_data private data for the slave irq handler.

private_data additional data, user customizable. Not touched by the MPU driver.

```
struct ext_slave_platform_data {
  __u8 type;
  __u32 irq;
  __u32 adapt_num;
  __u32 bus;
  __u8 address;
  __s8 orientation[9];
  void * irq_data;
  void * private_data;
};
```

6.10 inv_error_t Struct Reference

The MPL Error Code return type.

`#include "mltypes"`

6.10.1 Detailed Description

The MPL Error Code return type.

```
typedef unsigned char inv_error_t;
```

6.11 mpu_platform_data Struct Reference

Platform data for the mpu driver.

`#include <mpu.h>`

6.11.1 Detailed Description

Platform data for the mpu driver.

Contains platform specific information on how to configure the MPU3050 to work on this platform. The orientation matricies are 3x3 rotation matricies that are applied to the data to rotate from the mounting orientation to the platform orientation. The values must be one of 0, 1, or -1 and each row and column should have exactly 1 non-zero value.

Parameters:

int_config Bits [7:3] of the int config register.

level_shifter 0: VLogic, 1: VDD.

orientation[GYRO_NUM_AXES ∗GYRO_NUM_AXES] Orientation matrix of the gyroscope.

```
struct mpu_platform_data {
  __u8 int_config;
  __u8 level_shifter;
  __s8 orientation[GYRO_NUM_AXES  *GYRO_NUM_AXES];
};
```

6.12 tGesture Struct Reference

Gesture data structure.

6.12.1 Detailed Description

Gesture data structure.

When a gesture is detected a structure of this type is returned to inv_get_gesture() or the callback specified by inv_set_gesture_cb().

Parameters:

type Type of gesture. One of:

- INV_PITCH_SHAKE use tGestureShake
- INV_ROLL_SHAKE use tGestureShake
- INV_YAW_SHAKE use tGestureShake
- INV_TAP use tGestureTap
- INV_YAW_IMAGE_ROTATE use tGestureYawImageRotate

strength See tGestureShake, tGestureTap or tGestureYawImageRotate

speed See tGestureShake, tGestureTap or tGestureYawImageRotate

num See tGestureShake, tGestureTap or tGestureYawImageRotate

meta See tGestureShake, tGestureTap or tGestureYawImageRotate

reserved See tGestureShake, tGestureTap or tGestureYawImageRotate

```
typedef struct  {

    unsigned short type;
    short          strength;
    short          speed;
    unsigned short num;
    short          meta;
    short          reserved;

}   tGesture;
```

6.13 tGestureShake Struct Reference

Shake gesture data structure.

6.13.1 Detailed Description

Shake gesture data structure.

When a shake is detected a structure of this type is returned to inv_get_gesture() or the callback specified by inv_set_gesture_cb().

This structure contains the axis of the shake, and the number of shakes detected so far and the strength and speed of the shake.

Parameters:

type Type of gesture, set to one of

- INV_PITCH_SHAKE
- INV_ROLL_SHAKE
- INV_YAW_SHAKE

strength Type of shake. One of

- INV_SOFT_SHAKE
- INV_HARD_SHAKE

speed Maximum angular velocity of the shake or peak shake when multiple shakes have been detected. Units in degrees per second. For more precision use the reserved parameter.

num Number of Shakes detected so far. Use MLSetMaxShakdes() to set the maximum before this will be reset to 0.

meta Direction of the shake in the frame of reference of the device. Signed value with:

- 0: Positive
- Non-Zero: Negetive

reserved Fraction part of the maximum angular velocity of the shake or peak shake when multiple shakes have been detected. Units are 65536 lsb's per degree. The formula for creating a floating point representation of the speed is:

```
float speed = (float)gesture->speed +
              (float)(gesture->reserved/(float)(1 << 16));
```

6.14 tGestureTap Struct Reference

Tap gesture data structure.

6.14.1 Detailed Description

Tap gesture data structure.

When a tap is detected a structure of this type is returned to inv_get_gesture() or the callback specified by inv_set_gesture_cb().

If type field is INV_TAP then this structure contains the tap information including number of taps detected so far and the direction of the tap.

In addtion to the meta data telling the direction. The relative magnitude of the tap impulse on each axis is also specified in the following fields:

- strength: X
- speed: Y
- reserved: Z

Parameters:

type Type of gesture, set to INV_TAP for a tGestureTap type

strength Magnitude of the tap impulse on the X axis.

speed Magnitude of the tap impulse on the Y axis.

num Number of Taps detected so far. Use inv_set_max_taps() to set the maximum before this will be reset to 0.

meta Direction of the tap in the frame of reference of the device. Signed value with:

- 1: X
- 2: Y
- 3: Z

reserved Magnitude of the tap impulse on the Z axis.

6.15 tGestureYawImageRotate Struct Reference

Yaw image rotate gesture data structure.

6.15.1 Detailed Description

Yaw image rotate gesture data structure.

When a yaw image rotation is detected a structure of this type is returned to inv_get_gesture() or the callback specified by inv_set_gesture_cb().

This structure ccontains the direction of the Yaw Image Rotation.

Parameters:

type Type of gesture, set to INV_YAW_IMAGE_ROTATE

strength Unused.

speed Unused.

num Unused

meta Direction of the rotation in the frame of reference of the device. :

- 0: Positive
- Non-Zero: Negetive

reserved Unused

6.16 tMLGlyphData Struct Reference

Describes the data to be used by the character recognition algorithm.

`#include <mlglyph.h>`

6.16.1 Detailed Description

Describes the data to be used by the character recognition algorithm.

When training and recognizing characters, data is stored and read from this data container.

Parameters:

yGlyph

xGlyph

GlyphLen

features

gestures

segments

library

libraryLength

probs

finalGesture

updatingGlyph

speedThresh

probFinal

minProb

Index