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

Doc : SW-EMA-REL-5.1.1
Doc Rev : 1.0
Date : 11/16/2012

Embedded MotionApps v5.1.1 APIs Specification

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 Embedded MotionApps Platform Library (eMPL), and corresponds with Embedded MotionApps Release v5.1.1.

The eMPL contains the code for controlling the InvenSense devices, 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 eMPL are described in this document, including all parameters involved in the function calls. The functions are divided into modules as follows:

Module	Name	Description
Data Builder	Builds Sensor Data Structures	Builds the sensor structures and calls functions that need to use them.
HAL Outputs	HAL Outputs	Creates and holds information that a Android HAL layer might want.
Message Layer	Message Layer	Holds Messages
ML Math Func	Math Functions	Support Math Functions.
MPL	MPU Start	Handles init, start, and version properties .
Result_Holder	Result Holder	Holds various output results.
Start_Manager	Start Manager	Sends start events.
Storage_Manager	Store Variables	Stores Internal States.

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. The document is subdivided in the various sections, each describing the main source Modules composing the eMPL and implementing specific features.

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 **Embedded MotionApps 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

Module Documentation

4.1 accel_calibration

Accel calibration.

Files

- file accel_auto_cal.c
  Accel calibration.

Functions

- inv_error_t inv_disable_in_use_auto_calibration (void)
  Disables an algorithm to set accel biases.
- inv_error_t inv_enable_in_use_auto_calibration (void)
  Turns on an algorithm to set accel biases.
- inv_error_t inv_init_in_use_auto_calibration (void)
  Init in-use auto calibration.
- inv_error_t inv_start_in_use_auto_calibration (void)
  Start accel bias calibration.
- inv_error_t inv_stop_in_use_auto_calibration (void)
  Turns on an algorithm to set accel biases.

4.1.1 Detailed Description

Accel calibration.

4.1.2 Function Documentation

4.1.2.1 inv_error_t inv_disable_in_use_auto_calibration (void)

Disables an algorithm to set accel biases.

Typically called once per session. See inv_stop_in_use_auto_calibration() to stop the algorithm.

Returns:

INV_SUCCESS on success or an error code if call was not successful.

4.1.2.2 inv_error_t inv_enable_in_use_auto_calibration (void)

Turns on an algorithm to set accel biases.

This may be called after inv_init_mpl() and before inv_start_mpl(). It is typically only called once per session.

Returns:

INV_SUCCESS on success or an error code if call was not successful.

4.1.2.3 inv_error_t inv_start_in_use_auto_calibration (void)

Start accel bias calibration.

It is automatically called in start mode after an enable. This function only needs to be called to start after a stop command generated by inv_stop_in_use_auto_calibration().

Returns:

INV_SUCCESS on success or an error code if call was not successful.

4.1.2.4 inv_error_t inv_stop_in_use_auto_calibration (void)

Turns on an algorithm to set accel biases.

This may be called after inv_init_mpl() and before inv_start_mpl(). It is typically only called once per session. It does not return a motion state.

Returns:

INV_SUCCESS on success or an error code if call was not successful.

4.2 compass_vector_cal

A compass calibration algorithm that is mutually exclusive with compass_fit.

Files

- file compass_vec_cal.c

Functions

- inv_error_t inv_disable_vector_compass_cal (void)
  Disables a precise compass bias algorithm.
- inv_error_t inv_enable_vector_compass_cal (void)
  Enables a precise compass bias algorithm.
- inv_error_t inv_init_vector_compass_cal (void)
  Initializes/Resets this module.
- inv_error_t inv_start_vector_compass_cal (void)
  Allows the user to start a precise compass bias algorithm.
- inv_error_t inv_stop_vector_compass_cal (void)
  Allows the user to stop a precise compass bias algorithm.

4.2.1 Detailed Description

A compass calibration algorithm that is mutually exclusive with compass_fit.

4.2.2 Function Documentation

4.2.2.1 inv_error_t inv_disable_vector_compass_cal (void)

Disables a precise compass bias algorithm.

Should only be called once per library load when you wish to remove this functionality. See inv_stop_vector_compass_cal() if you wish to simply stop the algorithm.

Returns:

INV_SUCCESS on success or an error code if call was not successful.

4.2.2.2 inv_error_t inv_enable_vector_compass_cal (void)

Enables a precise compass bias algorithm.

This may be called after inv_init_mpl() and before inv_start_mpl(). It is typically only called once per session. It does not return a motion state. Mutually exclusive with inv_enable_compass_fit().

Returns:

INV_SUCCESS on success or an error code if call was not successful.

4.2.2.3 inv_error_t inv_init_vector_compass_cal (void)

Initializes/Resets this module.

Called by inv_enable_vector_compass_cal(). If you are calling this for testing, you probably also want to call inv_init_adv_fusion_obj()

Returns:

INV_SUCCESS on success or an error code if call was not successful.

4.2.2.4 inv_error_t inv_start_vector_compass_cal (void)

Allows the user to start a precise compass bias algorithm.

It is automatically called in start mode after an enable. This function only needs to be called to start after a stop command generated by inv_stop_vector_compass_cal().

Returns:

INV_SUCCESS on success or an error code if call was not successful.

4.2.2.5 inv_error_t inv_stop_vector_compass_cal (void)

Allows the user to stop a precise compass bias algorithm.

To start the algorithm back up call inv_start_vector_compass_cal()

Returns:

INV_SUCCESS on success or an error code if call was not successful.

4.3 fast_no_mot

Fast no motion algorithm used to set the gyro bias.

Files

- file fast_no_motion.c

  Fast no motion algorithm.

Functions

- void int_set_fast_nomot_gyro_threshold (long long thresh)

  Sets internal threshold for fast no motion.

- inv_error_t inv_disable_fast_nomot (void)

  Turns off a faster Motion/No Motion to set gyro biases (see inv_enable_fast_nomot()).

- inv_error_t inv_enable_fast_nomot (void)

  Turns on a faster Motion/No Motion to set gyro biases.

- void inv_fast_nomot_set_gyro_bias (struct inv_sensor_cal_t *sensor_cal)

  Used to set gyro bias when no motion is detected.

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

  This is used to help set inv_set_fast_nomot_accel_threshold().

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

  This is used to help set inv_set_fast_nomot_compass_threshold().

- long long inv_get_fnm_gyro_no_motion_param (void)

  Get gyro parameters.

- inv_error_t inv_init_fast_nomot (void)

  Initializes the fast no motion algorithm.

- void inv_set_default_number_of_samples (int count)

  Set default number of samples.

- void inv_set_fast_nomot_accel_threshold (long long thresh)

  Used to set internal threshold.

- void inv_set_fast_nomot_compass_threshold (long long thresh)
  Used to set internal threshold.
- inv_error_t inv_start_fast_nomot (void)
  Allows the user to start the fast no motion algorithm.
- inv_error_t inv_stop_fast_nomot (void)
  Allows the user to stop the fast no motion algorithm.

4.3.1 Detailed Description

Fast no motion algorithm used to set the gyro bias.

4.3.2 Function Documentation

4.3.2.1 inv_error_t inv_disable_fast_nomot (void)

Turns off a faster Motion/No Motion to set gyro biases (see inv_enable_fast_nomot()).

It is typically only called once per session. It does not return a motion state. It is mutually exclusive with inv_enable_motion_no_motion().

Returns:

INV_SUCCESS on success or an error code if call was not successful.

4.3.2.2 inv_error_t inv_enable_fast_nomot (void)

Turns on a faster Motion/No Motion to set gyro biases.

This may be called after inv_init_mpl() and before inv_start_mpl(). It is typically only called once per session. It does not return a motion state. It is mutually exclusive with inv_enable_motion_no_motion().

Returns:

INV_SUCCESS on success or an error code if call was not successful.

4.3.2.3 void inv_fast_nomot_set_gyro_bias (struct inv_sensor_cal_t * *sensor_cal*)

Used to set gyro bias when no motion is detected.

Parameters:

sensor_cal,: pointer of the sensor data strurcture

4.3.2.4 void inv_get_fast_nomot_accel_param (long * *cntr*, long long * *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 parameter for inv_set_fast_nomot_accel_threshold(). The compass must be on.

Parameters:

cntr Counter for when param changes

param Parameter used to help set threshold

4.3.2.5 void inv_get_fast_nomot_compass_param (long * *cntr*, long long * *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.

Parameters:

cntr Counter for when param changes

param Parameter used to help set threshold

4.3.2.6 inv_error_t inv_init_fast_nomot (void)

Initializes the fast no motion algorithm.

Automatically called by inv_enable_fast_nomot(). Not typically called by the user.

Returns:

INV_SUCCESS on success or an error code if call was not successful.

4.3.2.7 void inv_set_default_number_of_samples (int *count*)

Set default number of samples.

Not typically called by users.

Parameters:

count Number of samples to use for algorithm

4.3.2.8 void inv_set_fast_nomot_accel_threshold (long long *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

4.3.2.9 void inv_set_fast_nomot_compass_threshold (long long *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

4.3.2.10 inv_error_t inv_start_fast_nomot (void)

Allows the user to start the fast no motion algorithm.

It is automatically in start mode after an enable. This function only needs to be called to start after a stop command generated by inv_stop_fast_nomot().

Returns:

INV_SUCCESS on success or an error code if call was not successful.

4.3.2.11 inv_error_t inv_stop_fast_nomot (void)

Allows the user to stop the fast no motion algorithm.

See inv_start_fast_nomot() to start the algorithm back up.

Returns:

INV_SUCCESS on success or an error code if call was not successful.

4.4 nine_axis_fusion

Performs nine axis sensor fusion.

Files

- file fusion_9axis.c

  Performs nine axis sensor fusion.

Functions

- inv_error_t inv_9x_fusion_enable_jitter_reduction (int en)

  This enables the jitter reduction feature.

- inv_error_t inv_9x_fusion_set_mag_fb (float fb)

  This sets the magnetic feedback.

- inv_error_t inv_9x_fusion_use_timestamps (int en)

  Use timestamps when evaluating compass correction gain.

- inv_error_t inv_disable_9x_sensor_fusion ()

  Disables the 9 axis sensor fusion algorithm.

- inv_error_t inv_enable_9x_sensor_fusion (void)

  Enables the 9 axis sensor fusion algorithm.

- void inv_init_9x_fusion (void)

  Initializes the algorithm.

- inv_error_t inv_start_9x_sensor_fusion (void)

  Starts the 9 axis sensor fusion.

- inv_error_t inv_stop_9x_sensor_fusion (void)

  Stops the 9 axis sensor fusion from running.

4.4.1 Detailed Description

Performs nine axis sensor fusion.

4.4.2 Function Documentation

4.4.2.1 inv_error_t inv_9x_fusion_enable_jitter_reduction (int *en*)

This enables the jitter reduction feature.

Parameters:

en Should be non-zero to enable the feature. Initialized to 0, i.e. off

Returns:

heading correction angle

4.4.2.2 inv_error_t inv_9x_fusion_set_mag_fb (float *fb*)

This sets the magnetic feedback.

Increasing it resutls in faster compass correction in the 9 axis quaternion.

Parameters:

fb Desired magnetic feedback value. Typical value is 1. Also, initialized to 1 in inv_init_9x_fusion.

Returns:

heading correction angle

4.4.2.3 inv_error_t inv_9x_fusion_use_timestamps (int *en*)

Use timestamps when evaluating compass correction gain.

This feature should be used when the MPL is not receiving compass data at a constant rate.

Parameters:

en 1 to enable the feature.

Returns:

INV_SUCCESS on success or an error code if call was not successful.

4.4.2.4 inv_error_t inv_disable_9x_sensor_fusion ()

Disables the 9 axis sensor fusion algorithm.

Should only be called once per library load when you wish to remove this functionality. See inv_stop_9x_sensor_fusion() if you wish to simply stop the algorithm.

Returns:

INV_SUCCESS on success or an error code if call was not successful.

4.4.2.5 inv_error_t inv_enable_9x_sensor_fusion (void)

Enables the 9 axis sensor fusion algorithm.

This should only be called once per library load. See inv_start_9x_sensor_fusion() and inv_stop_9x_sensor_fusion() for starting and stopping. Automatically calls inv_start_9x_sensor_fusion() and inv_init_9x_fusion().

Returns:

INV_SUCCESS on success or an error code if call was not successful.

4.4.2.6 void inv_init_9x_fusion (void)

Initializes the algorithm.

Automatically called by inv_enable_9x_sensor_fusion(). Not normally called by users.

4.4.2.7 inv_error_t inv_start_9x_sensor_fusion (void)

Starts the 9 axis sensor fusion.

Automatically called by inv_enable_9x_sensor_fusion() and only needs to be called after stopping with inv_stop_9x_sensor_fusion().

Returns:

INV_SUCCESS on success or an error code if call was not successful.

4.4.2.8 inv_error_t inv_stop_9x_sensor_fusion (void)

Stops the 9 axis sensor fusion from running.

See inv_start_9x_sensor_fusion() to start it back up again.

Returns:

INV_SUCCESS on success or an error code if call was not successful.

4.5 gyro_tc

Gyro Temperature Compensation algorithm.

Files

- file gyro_tc.c
  Gyro bias temperature compensation.

Functions

- inv_error_t inv_disable_gyro_tc (void)
  Enable the gyro temp comp algorithm.
- inv_error_t inv_enable_gyro_tc (void)
  Enable the gyro temp comp algorithm.
- inv_error_t inv_init_gyro_ts (void)
  Reset the gyro temp slope.
- inv_error_t inv_start_gyro_tc (void)
  Registers callback to receive new temperature data.
- inv_error_t inv_stop_gyro_tc (void)
  Unregisters callback.

4.5.1 Detailed Description

Gyro Temperature Compensation algorithm.

4.5.2 Function Documentation

4.5.2.1 inv_error_t inv_disable_gyro_tc (void)

Enable the gyro temp comp algorithm.

Returns:

INV_SUCCESS if successful.

4.5.2.2 inv_error_t inv_enable_gyro_tc (void)

Enable the gyro temp comp algorithm.

Returns:

INV_SUCCESS if successful.

4.5.2.3 inv_error_t inv_init_gyro_ts (void)

Reset the gyro temp slope.

Returns:

INV_SUCCESS if successful.

4.5.2.4 inv_error_t inv_start_gyro_tc (void)

Registers callback to receive new temperature data.

Returns:

INV_SUCCESS if successful.

4.5.2.5 inv_error_t inv_stop_gyro_tc (void)

Unregisters callback.

Returns:

INV_SUCCESS if successful.

4.6 heading_from_gyro

A less accurate but fast algorithm for 9 axis sensor fusion.

Files

- file heading_from_gyro.c

Functions

- inv_error_t inv_disable_heading_from_gyro (void)

  Turns off a heading from gyro.

- inv_error_t inv_enable_heading_from_gyro (void)

  Turns on a heading from gyro algorithm which performs sensor fusion when the compass bias hasn't been fully solved for.

- void inv_init_heading_from_gyro (void)

  Initializes/Resets this module.

- inv_error_t inv_start_heading_from_gyro (void)

  Registers callback to recieve gyro and compass data.

- inv_error_t inv_stop_heading_from_gyro (void)

  Unregisters callback.

4.6.1 Detailed Description

A less accurate but fast algorithm for 9 axis sensor fusion.

4.6.2 Function Documentation

4.6.2.1 inv_error_t inv_disable_heading_from_gyro (void)

Turns off a heading from gyro.

It is typically only called once per session.

Returns:

INV_SUCCESS if successful.

4.6.2.2 inv_error_t inv_enable_heading_from_gyro (void)

Turns on a heading from gyro algorithm which performs sensor fusion when the compass bias hasn't been fully solved for.

This may be called after inv_init_mpl() and before inv_start_mpl(). It is typically only called once per session.

Returns:

INV_SUCCESS if successful.

4.6.2.3 void inv_init_heading_from_gyro (void)

Initializes/Resets this module.

Called by inv_enable_heading_from_gyro().

Returns:

INV_SUCCESS if successful.

4.6.2.4 inv_error_t inv_start_heading_from_gyro (void)

Registers callback to recieve gyro and compass data.

Returns:

INV_SUCCESS if successful.

4.6.2.5 inv_error_t inv_stop_heading_from_gyro (void)

Unregisters callback.

Returns:

INV_SUCCESS if successful.

4.7 mag_disturb

Determines magnetic disturbances and sets compass accuracy appropriately.

Files

- file mag_disturb.c

Functions

- inv_error_t inv_disable_magnetic_disturbance (void)

  Turns off a magnetic disturbance algorithm (see inv_enable_magnetic_disturbance()).

- inv_error_t inv_enable_magnetic_disturbance (void)

  Enables a magnetic disturbance algorithm.

- inv_error_t inv_start_magnetic_disturbance (void)

  Allows the user to start the magnetic disturbance algorithm.

- inv_error_t inv_stop_magnetic_disturbance (void)

  Allows the user to stop the magnetic disturbance algorithm.

4.7.1 Detailed Description

Determines magnetic disturbances and sets compass accuracy appropriately.

4.7.2 Function Documentation

4.7.2.1 inv_error_t inv_disable_magnetic_disturbance (void)

Turns off a magnetic disturbance algorithm (see inv_enable_magnetic_disturbance()).

It is typically only called once per session. See inv_stop_magnetic_disturbance() to stop the algorithm

Returns:

INV_SUCCESS on success or an error code if call was not successful.

4.7.2.2 inv_error_t inv_enable_magnetic_disturbance (void)

Enables a magnetic disturbance algorithm.

This may be called after inv_init_mpl() and before inv_start_mpl(). It is typically only called once per session. It does not return a motion state.

Returns:

INV_SUCCESS on success or an error code if call was not successful.

4.7.2.3 inv_error_t inv_start_magnetic_disturbance (void)

Allows the user to start the magnetic disturbance algorithm.

It is automatically called in start mode after an enable. This function only needs to be called to start after a stop command generated by inv_stop_magnetic_disturbance().

Returns:

INV_SUCCESS on success or an error code if call was not successful.

4.7.2.4 inv_error_t inv_stop_magnetic_disturbance (void)

Allows the user to stop the magnetic disturbance algorithm.

To start the algorithm back up call inv_start_no_gyro_fusion()

Returns:

INV_SUCCESS on success or an error code if call was not successful.

4.8 motion_no_motion

A motion detection algorithm that is used to set gyro bias when the device is not moving.

Files

- file motion_no_motion.c

  A motion detection algorithm that is used to set gyro bias when the device is not moving.

Functions

- inv_error_t inv_disable_motion_no_motion (void)

  Turns off Motion/No Motion to set gyro biases (see inv_enable_motion_no_motion()).

- inv_error_t inv_enable_motion_no_motion ()

  Turns on Motion/No Motion used to set gyro biases.

- inv_error_t inv_init_motion_no_motion (void)

  Initializes the motion no motion algorithm.

- inv_error_t inv_set_no_motion_time (long time_ms)

  Allows the user to set the time to be in a no motion state before setting the gyro bias.

- inv_error_t inv_start_motion_no_motion (void)

  Allows the user to start the no motion algorithm.

- inv_error_t inv_stop_motion_no_motion (void)

  Allows the user to stop the no motion algorithm.

4.8.1 Detailed Description

A motion detection algorithm that is used to set gyro bias when the device is not moving.

4.8.2 Function Documentation

4.8.2.1 inv_error_t inv_disable_motion_no_motion (void)

Turns off Motion/No Motion to set gyro biases (see inv_enable_motion_no_motion()).

It is typically only called once per session. It does not return a motion state. It is mutually exclusive with inv_enable_fast_nomot().

Returns:

INV_SUCCESS on success or an error code if call was not successful.

4.8.2.2 inv_error_t inv_enable_motion_no_motion ()

Turns on Motion/No Motion used to set gyro biases.

This may be called after inv_init_mpl() and before inv_start_mpl(). It is typically only called once per session. It does not return a motion state. It is mutually exclusive with inv_enable_motion_no_motion().

Returns:

INV_SUCCESS on success or an error code if call was not successful.

4.8.2.3 inv_error_t inv_init_motion_no_motion (void)

Initializes the motion no motion algorithm.

Automatically called by inv_enable_motion_no_motion(). Not typically called by the user.

Returns:

INV_SUCCESS on success or an error code if call was not successful.

4.8.2.4 inv_error_t inv_set_no_motion_time (long *time_ms*)

Allows the user to set the time to be in a no motion state before setting the gyro bias.

Parameters:

time_ms Time in milliseconds. Default is 8000ms or 8 seconds.

Returns:

INV_SUCCESS on success or an error code if call was not successful.

4.8.2.5 inv_error_t inv_start_motion_no_motion (void)

Allows the user to start the no motion algorithm.

It is automatically called in start mode after an enable. This function only needs to be called to start after a stop command generated by inv_stop_motion_no_motion().

Returns:

INV_SUCCESS on success or an error code if call was not successful.

4.8.2.6 inv_error_t inv_stop_motion_no_motion (void)

Allows the user to stop the no motion algorithm.

See inv_start_motion_no_motion() to start the algorithm back up.

Returns:

INV_SUCCESS on success or an error code if call was not successful.

4.9 no_gyro_fusion

Accel/Compass Sensor fusion.

Files

- file no_gyro_fusion.c

  Accel/Compass Sensor fusion.

Functions

- inv_error_t inv_disable_no_gyro_fusion (void)

  Turns off a sensor fusion using accel and compass only (see inv_enable_no_gyro_fusion()).

- inv_error_t inv_enable_no_gyro_fusion (void)

  Enables a sensor fusion using accel and compass only.

- inv_error_t inv_init_no_gyro_fusion (void)

  Initializes the algorithm.

- inv_error_t inv_start_no_gyro_fusion (void)

  Allows the user to start the sensor fusion using accel and compass only algorithm.

- inv_error_t inv_stop_no_gyro_fusion (void)

  Allows the user to stop the sensor fusion using accel and compass only algorithm.

4.9.1 Detailed Description

Accel/Compass Sensor fusion.

4.9.2 Function Documentation

4.9.2.1 inv_error_t inv_disable_no_gyro_fusion (void)

Turns off a sensor fusion using accel and compass only (see inv_enable_no_gyro_fusion()).

It is typically only called once per session. See inv_stop_no_gyro_fusion() to stop the algorithm

Returns:

INV_SUCCESS on success or an error code if call was not successful.

4.9.2.2 inv_error_t inv_enable_no_gyro_fusion (void)

Enables a sensor fusion using accel and compass only.

This may be called after inv_init_mpl() and before inv_start_mpl(). It is typically only called once per session. It does not return a motion state.

Returns:

INV_SUCCESS on success or an error code if call was not successful.

4.9.2.3 inv_error_t inv_init_no_gyro_fusion (void)

Initializes the algorithm.

Automatically called by the enable function.

Returns:

INV_SUCCESS on success or an error code if call was not successful.

4.9.2.4 inv_error_t inv_start_no_gyro_fusion (void)

Allows the user to start the sensor fusion using accel and compass only algorithm.

It is automatically called in start mode after an enable. This function only needs to be called to start after a stop command generated by inv_stop_no_gyro_fusion().

Returns:

INV_SUCCESS on success or an error code if call was not successful.

4.9.2.5 inv_error_t inv_stop_no_gyro_fusion (void)

Allows the user to stop the sensor fusion using accel and compass only algorithm.

See inv_start_no_gyro_fusion() to start the algorithm back up call inv_start_no_gyro_-fusion()

Returns:

INV_SUCCESS on success or an error code if call was not successful.

4.10 quaternion_supervisor

Motion Library - Generates the 6-axis quaternion.

Files

- file quaternion_supervisor.c

  Performs the quaternion fusion.

Functions

- inv_error_t inv_disable_quaternion (void)

  Disables generating the gyro and accel quaternion.

- inv_error_t inv_enable_quaternion ()

  Turns on quaternion computation.

- inv_error_t inv_init_quaternion (void)

  Initializes all quaternion data.

- void inv_set_quaternion (long *quat)

  Set the quaternion to the given value.

- inv_error_t inv_start_quaternion (void)

  Starts gyro and accel quaternion generation.

- inv_error_t inv_stop_quaternion (void)

  Stops gyro and accel quaternion generation.

4.10.1 Detailed Description

Motion Library - Generates the 6-axis quaternion.

4.10.2 Function Documentation

4.10.2.1 inv_error_t inv_enable_quaternion ()

Turns on quaternion computation.

This must be called after inv_init_mpl() and before inv_start_mpl(). It is typically only called once per session. inv_start_quaterion() and inv_stop_quaternion() are used to start and stop this feature. This feature is started automatically and inv_start_-quaterion() would only need to be called after turning this feature off with inv_stop_-quaternion().

Returns:

INV_SUCCESS=0 on success, a non-zero error code otherwise.

4.10.2.2 inv_error_t inv_init_quaternion (void)

Initializes all quaternion data.

This is called automatically by the enable function. It may be called any time the feature is enabled, but is typically not needed to be called by outside callers.

Returns:

INV_SUCCESS=0 on success, a non-zero error code otherwise.

4.10.2.3 void inv_set_quaternion (long * *quat*)

Set the quaternion to the given value.

Parameters:

quat What to set quaternion to. Fixed point scaled by 2^30, Length 4.

4.10.2.4 inv_error_t inv_start_quaternion (void)

Starts gyro and accel quaternion generation.

Automatically called by inv_enable_quaterion() and therefor would only need to be called after inv_stop_quaternion().

Returns:

INV_SUCCESS=0 on success, a non-zero error code otherwise.

4.10.2.5 inv_error_t inv_stop_quaternion (void)

Stops gyro and accel quaternion generation.

Call inv_start_quaternion() to turn this back on after the stop command.

Returns:

INV_SUCCESS=0 on success, a non-zero error code otherwise.

4.11 data_builder

Motion Library - Data Builder Constructs and Creates the data for MPL.

Files

- file data_builder.c

  Data Builder.

Defines

- #define INV_DB_SAVE_KEY 53395

  Change this key if the data being stored by this file changes.

Functions

- void inv_accel_was_turned_off ()

  This should be called when the accel has been turned off.

- void inv_apply_calibration (struct inv_single_sensor_t *sensor, const long *bias)

  Takes raw data stored in the sensor, removes bias, and converts it to calibrated data in the body frame.

- inv_error_t inv_build_accel (const long *accel, int status, inv_time_t timestamp)

  Record new accel data for use when inv_execute_on_data() is called.

- inv_error_t inv_build_compass (const long *compass, int status, inv_time_t timestamp)

  Record new compass data for use when inv_execute_on_data() is called.

- inv_error_t inv_build_gyro (const short *gyro, inv_time_t timestamp)

  Record new gyro data and calls inv_execute_on_data() if previous sample has not been processed.

- inv_error_t inv_build_quat (const long *quat, int status, inv_time_t timestamp)

  quaternion data

- inv_error_t inv_build_temp (const long temp, inv_time_t timestamp)

Record new temperature data for use when inv_execute_on_data() is called.

- void inv_compass_was_turned_off ()

  This should be called when the compass has been turned off.

- void inv_disable_compass_soft_iron_matrix (void)

  This subroutine disables the the soft iron transformation process.

- void inv_enable_compass_soft_iron_matrix (void)

  This subroutine enables the the soft iron transformation process.

- inv_error_t inv_execute_on_data (void)

  After at least one of inv_build_gyro(), inv_build_accel(), or inv_build_compass() has been called, this function should be called.

- int inv_get_accel_accuracy (void)

  Returns accuracy of accel.

- void inv_get_accel_bias (long *bias, long *temp)

  Get Accel Bias.

- int inv_get_accel_on ()

  Helper function stating whether the acceleromter is on or off.

- long inv_get_accel_sensitivity (void)

  Accel sensitivity.

- void inv_get_accel_set (long *data, int8_t *accuracy, inv_time_t *timestamp)

  Gets a whole set of accel data including data, accuracy and timestamp.

- void inv_get_compass_bias (long *bias)

  Returns the current bias for the compass.

- int inv_get_compass_on ()

  Helper function stating whether the compass is on or off.

- long inv_get_compass_sensitivity (void)

  Compass sensitivity.

- void inv_get_compass_set (long *data, int8_t *accuracy, inv_time_t *timestamp)

  Gets a whole set of compass data including data, accuracy and timestamp.

- void inv_get_compass_soft_iron_input_data (long *data)

  This subroutine gets the fixed point Q30 compass data before the soft iron transformation.

- void inv_get_compass_soft_iron_matrix_d (long *matrix)

  Gets the 3x3 compass transform matrix in 32 bit Q30 fixed point format.

- void inv_get_compass_soft_iron_matrix_f (float *matrix)

  Gets the 3x3 compass transform matrix in 32 bit floating point format.

- void inv_get_compass_soft_iron_output_data (long *data)

  This subroutine gets the fixed point Q30 compass data after the soft iron transformation.

- void inv_get_gyro (long *gyro)

  Get's latest gyro data.

- int inv_get_gyro_accuracy (void)

  Returns accuracy of gyro.

- void inv_get_gyro_bias (long *bias, long *temp)

  Get the gyro biases and temperature record from MPL.

- int inv_get_gyro_on ()

  Helper function stating whether the gyro is on or off.

- long inv_get_gyro_sensitivity ()

  Gyro sensitivity.

- void inv_get_gyro_set (long *data, int8_t *accuracy, inv_time_t *timestamp)

  Gets a whole set of gyro data including data, accuracy and timestamp.

- void inv_get_gyro_set_raw (long *data, int8_t *accuracy, inv_time_t *timestamp)

  Gets a whole set of gyro raw data including data, accuracy and timestamp.

- inv_time_t inv_get_last_timestamp ()

  Get last timestamp across all 3 sensors that are on.

- int inv_get_mag_accuracy (void)

  Returns accuracy of compass.

- void inv_get_temp_set (long *data, int *accuracy, inv_time_t *timestamp)

Gets a whole set of temperature data including data, accuracy and timestamp.

- void inv_gyro_was_turned_off ()

  This should be called when the gyro has been turned off.

- inv_error_t inv_init_data_builder (void)

  Initialize the data builder.

- void inv_quaternion_sensor_was_turned_off (void)

  This should be called when the quaternion data from the DMP has been turned off.

- inv_error_t inv_register_data_cb (inv_error_t(∗func)(struct inv_sensor_cal_-t ∗data), int priority, int sensor_type)

  Registers to receive a callback when there is new sensor data.

- void inv_reset_compass_soft_iron_matrix (void)

  This subroutine resets the the soft iron transformation to unity matrix and disable the soft iron transformation process by default.

- void inv_set_accel_accuracy (int accuracy)

  Sets the accel accuracy.

- void inv_set_accel_bandwidth (int bandwidth_hz)

  Set Accel Bandwidth in Hz.

- void inv_set_accel_bias (const long ∗bias, int accuracy)

  Sets the accel bias.

- void inv_set_accel_bias_mask (const long ∗bias, int accuracy, int mask)

  Sets the accel bias with control over which axis.

- void inv_set_accel_orientation_and_scale (int orientation, long sensitivity)

  Sets the orientation and sensitivity of the gyro data.

- void inv_set_accel_sample_rate (long sample_rate_us)

  Set Accel Sample rate in micro seconds.

- void inv_set_compass_bandwidth (int bandwidth_hz)

  Set Compass Bandwidth in Hz.

- void inv_set_compass_disturbance (int dist)

  Set the state of a compass disturbance.

- void inv_set_compass_orientation_and_scale (int orientation, long sensitivity)

  Sets the Orientation and Sensitivity of the gyro data.

- void inv_set_compass_sample_rate (long sample_rate_us)

  Set Compass Sample rate in micro seconds.

- void inv_set_compass_soft_iron_input_data (const long *data)

  This subroutine sets the compass raw data for the soft iron transformation.

- void inv_set_compass_soft_iron_matrix_d (long *matrix)

  Sets the 3x3 compass transform matrix in 32 bit Q30 fixed point format.

- void inv_set_compass_soft_iron_matrix_f (float *matrix)

  Sets the 3x3 compass transform matrix in 32 bit floating point format.

- void inv_set_gyro_bandwidth (int bandwidth_hz)

  Set Gyro Bandwidth in Hz.

- void inv_set_gyro_bias (const long *bias, int accuracy)

  Sets the gyro bias.

- void inv_set_gyro_orientation_and_scale (int orientation, long sensitivity)

  Sets the Orientation and Sensitivity of the gyro data.

- void inv_set_gyro_sample_rate (long sample_rate_us)

  Set Gyro Sample rate in micro seconds.

- void inv_set_quat_sample_rate (long sample_rate_us)

  Set Quat Sample rate in micro seconds.

- void inv_temperature_was_turned_off ()

  This should be called when the temperature sensor has been turned off.

- inv_error_t inv_unregister_data_cb (inv_error_t(*func)(struct inv_sensor_cal_t *data))

  Unregisters the callback that happens when new sensor data is received.

- void set_sensor_orientation_and_scale (struct inv_single_sensor_t *sensor, int orientation, long sensitivity)

  Sets orientation and sensitivity field for a sensor.

4.11.1 Detailed Description

Motion Library - Data Builder Constructs and Creates the data for MPL.

4.11.2 Function Documentation

4.11.2.1 void inv_accel_was_turned_off ()

This should be called when the accel has been turned off.

This is so that we will know if the data is contiguous.

4.11.2.2 void inv_apply_calibration (struct inv_single_sensor_t * *sensor*, const long * *bias*)

Takes raw data stored in the sensor, removes bias, and converts it to calibrated data in the body frame.

Also store raw data for body frame.

Parameters:

sensor structure to modify

bias bias in the mounting frame, in hardware units scaled by 2^16. Length 3.

4.11.2.3 inv_error_t inv_build_accel (const long * *accel*, int *status*, inv_time_t *timestamp*)

Record new accel data for use when inv_execute_on_data() is called.

Parameters:

accel accel data. Length 3. Calibrated data is in m/s^2 scaled by 2^16 in body frame. Raw data is in device units in chip mounting frame.

status Lower 2 bits are the accuracy, with 0 being inaccurate, and 3 being most accurate. The upper bit INV_CALIBRATED, is set if the data was calibrated outside MPL and it is not set if the data being passed is raw. Raw data should be in device units, typically in a 16-bit range.

timestamp Monotonic time stamp, for Android it's in nanoseconds.

Returns:

Returns INV_SUCCESS if successful or an error code if not.

4.11.2.4 inv_error_t inv_build_compass (const long * *compass*, int *status*, inv_time_t *timestamp*)

Record new compass data for use when inv_execute_on_data() is called.

Parameters:

compass Compass data, if it was calibrated outside MPL, the units are uT scaled by 2^16. Length 3.

status Lower 2 bits are the accuracy, with 0 being inaccurate, and 3 being most accurate. The upper bit INV_CALIBRATED, is set if the data was calibrated outside MPL and it is not set if the data being passed is raw. Raw data should be in device units, typically in a 16-bit range.

timestamp Monotonic time stamp, for Android it's in nanoseconds.

executed Set to 1 if data processing was done.

Returns:

Returns INV_SUCCESS if successful or an error code if not.

4.11.2.5 inv_error_t inv_build_gyro (const short * *gyro*, inv_time_t *timestamp*)

Record new gyro data and calls inv_execute_on_data() if previous sample has not been processed.

Parameters:

gyro Data is in device units. Length 3.

timestamp Monotonic time stamp, for Android it's in nanoseconds.

executed Set to 1 if data processing was done.

Returns:

Returns INV_SUCCESS if successful or an error code if not.

4.11.2.6 inv_error_t inv_build_quat (const long * *quat*, int *status*, inv_time_t *timestamp*)

quaternion data

Parameters:

quat Quaternion data. 2^30 = 1.0 or 2^14=1 for 16-bit data. Real part first. Length 4.

status number of axis, 16-bit or 32-bit

timestamp

timestamp Monotonic time stamp; for Android it's in nanoseconds.

executed Set to 1 if data processing was done.

Returns:

Returns INV_SUCCESS if successful or an error code if not.

4.11.2.7 inv_error_t inv_build_temp (const long *temp*, inv_time_t *timestamp*)

Record new temperature data for use when inv_execute_on_data() is called.

Parameters:

temp Temperature data in q16 format.

timestamp Monotonic time stamp; for Android it's in nanoseconds.

Returns:

Returns INV_SUCCESS if successful or an error code if not.

4.11.2.8 void inv_compass_was_turned_off ()

This should be called when the compass has been turned off.

This is so that we will know if the data is contiguous.

4.11.2.9 inv_error_t inv_execute_on_data (void)

After at least one of inv_build_gyro(), inv_build_accel(), or inv_build_compass() has been called, this function should be called.

It will process the data it has received and update all the internal states and features that have been turned on.

Returns:

Returns INV_SUCCESS if successful or an error code if not.

4.11.2.10 int inv_get_accel_accuracy (void)

Returns accuracy of accel.

Returns:

Accuracy of accel with 0 being not accurate, and 3 being most accurate.

4.11.2.11 void inv_get_accel_bias (long * *bias*, long * *temp*)

Get Accel Bias.

Parameters:

bias Accel bias where

temp Temperature where 1 C = 2^16

4.11.2.12 int inv_get_accel_on ()

Helper function stating whether the acceleromter is on or off.

Returns:

TRUE if accel if on, 0 if accel if off

4.11.2.13 long inv_get_accel_sensitivity (void)

Accel sensitivity.

Returns:

A scale factor to convert device units to g's scaled by 2^16 such that g_s = device_-units * sensitivity / 2^30. Typically it works out to be the maximum accel value in g's * 2^15.

4.11.2.14 void inv_get_accel_set (long * *data*, int8_t * *accuracy*, inv_time_t * *timestamp*)

Gets a whole set of accel data including data, accuracy and timestamp.

Parameters:

data Accel Data where 1g = 2^16

accuracy Accuracy 0 being not accurate, and 3 being most accurate.

timestamp The timestamp of the data sample.

4.11.2.15 void inv_get_compass_bias (long * *bias*)

Returns the current bias for the compass.

Parameters:

bias Compass bias in hardware units scaled by 2^16. In mounting frame. Length 3.

4.11.2.16 int inv_get_compass_on ()

Helper function stating whether the compass is on or off.

Returns:

TRUE if compass if on, 0 if compass if off

4.11.2.17 long inv_get_compass_sensitivity (void)

Compass sensitivity.

Returns:

A scale factor to convert device units to micro Tesla scaled by 2^16 such that uT = device_units * sensitivity / 2^30. Typically it works out to be the maximum uT * 2^15.

4.11.2.18 void inv_get_compass_set (long * *data*, int8_t * *accuracy*, inv_time_t * *timestamp*)

Gets a whole set of compass data including data, accuracy and timestamp.

Parameters:

data Compass Data where 1 uT = 2^16

accuracy Accuracy 0 being not accurate, and 3 being most accurate.

timestamp The timestamp of the data sample.

4.11.2.19 void inv_get_compass_soft_iron_input_data (long * *data*)

This subroutine gets the fixed point Q30 compass data before the soft iron transformation.

Parameters:

the pointer of the 3x1 vector compass data in MPL format

4.11.2.20 void inv_get_compass_soft_iron_matrix_d (long * *matrix*)

Gets the 3x3 compass transform matrix in 32 bit Q30 fixed point format.

Parameters:

the pointer of the 3x3 matrix in Q30 format

4.11.2.21 void inv_get_compass_soft_iron_matrix_f (float * *matrix*)

Gets the 3x3 compass transform matrix in 32 bit floating point format.

Parameters:

the pointer of the 3x3 matrix in floating point format

4.11.2.22 void inv_get_compass_soft_iron_output_data (long * *data*)

This subroutine gets the fixed point Q30 compass data after the soft iron transformation.

Parameters:

the pointer of the 3x1 vector compass data in MPL format

4.11.2.23 void inv_get_gyro (long * *gyro*)

Get's latest gyro data.

Parameters:

gyro Gyro Data, Length 3. 1 dps = 2^16.

4.11.2.24 int inv_get_gyro_accuracy (void)

Returns accuracy of gyro.

Returns:

Accuracy of gyro with 0 being not accurate, and 3 being most accurate.

4.11.2.25 void inv_get_gyro_bias (long * *bias*, long * *temp*)

Get the gyro biases and temperature record from MPL.

Parameters:

bias Gyro bias in hardware units scaled by 2^16. In chip mounting frame. Length 3.

temp Tempearature in degrees C.

4.11.2.26 int inv_get_gyro_on ()

Helper function stating whether the gyro is on or off.

Returns:

TRUE if gyro if on, 0 if gyro if off

4.11.2.27 long inv_get_gyro_sensitivity ()

Gyro sensitivity.

Returns:

A scale factor to convert device units to degrees per second scaled by 2^16 such that degrees_per_second = device_units * sensitivity / 2^30. Typically it works out to be the maximum rate * 2^15.

4.11.2.28 void inv_get_gyro_set (long * *data*, int8_t * *accuracy*, inv_time_t * *timestamp*)

Gets a whole set of gyro data including data, accuracy and timestamp.

Parameters:

data Gyro Data where 1 dps = 2^16

accuracy Accuracy 0 being not accurate, and 3 being most accurate.

timestamp The timestamp of the data sample.

4.11.2.29 void inv_get_gyro_set_raw (long * *data*, int8_t * *accuracy*, inv_time_t * *timestamp*)

Gets a whole set of gyro raw data including data, accuracy and timestamp.

Parameters:

data Gyro Data where 1 dps = 2^16

accuracy Accuracy 0 being not accurate, and 3 being most accurate.

timestamp The timestamp of the data sample.

4.11.2.30 inv_time_t inv_get_last_timestamp ()

Get last timestamp across all 3 sensors that are on.

This find out which timestamp has the largest value for sensors that are on.

Returns:

Returns INV_SUCCESS if successful or an error code if not.

4.11.2.31 int inv_get_mag_accuracy (void)

Returns accuracy of compass.

Returns:

Accuracy of compass with 0 being not accurate, and 3 being most accurate.

4.11.2.32 void inv_get_temp_set (long * *data*, int * *accuracy*, inv_time_t * *timestamp*)

Gets a whole set of temperature data including data, accuracy and timestamp.

Parameters:

data Temperature data where 1 degree C = 2^16

accuracy 0 to 3, where 3 is most accurate.

timestamp The timestamp of the data sample.

4.11.2.33 void inv_gyro_was_turned_off ()

This should be called when the gyro has been turned off.

This is so that we will know if the data is contiguous.

4.11.2.34 void inv_quaternion_sensor_was_turned_off (void)

This should be called when the quaternion data from the DMP has been turned off.

This is so that we will know if the data is contiguous.

4.11.2.35 void inv_set_accel_accuracy (int *accuracy*)

Sets the accel accuracy.

Parameters:

accuracy Accuracy rating from 0 to 3, with 3 being most accurate.

4.11.2.36 void inv_set_accel_bandwidth (int *bandwidth_hz*)

Set Accel Bandwidth in Hz.

Parameters:

bandwidth_hz Gyro bandwidth in Hz

4.11.2.37 void inv_set_accel_bias (const long * *bias*, int *accuracy*)

Sets the accel bias.

Parameters:

bias Accel bias, length 3. In HW units scaled by 2^16 in body frame

accuracy Accuracy rating from 0 to 3, with 3 being most accurate.

4.11.2.38 void inv_set_accel_bias_mask (const long * *bias*, int *accuracy*, int *mask*)

Sets the accel bias with control over which axis.

Parameters:

bias Accel bias, length 3. In HW units scaled by 2^16 in body frame

accuracy Accuracy rating from 0 to 3, with 3 being most accurate.

mask Mask to select axis to apply bias set.

4.11.2.39 void inv_set_accel_orientation_and_scale (int *orientation*, long *sensitivity*)

Sets the orientation and sensitivity of the gyro data.

Parameters:

orientation A scalar defining the transformation from chip mounting to the body frame. The function inv_orientation_matrix_to_scalar() can convert the transformation matrix to this scalar and describes the scalar in further detail.

sensitivity A scale factor to convert device units to g's such that g's = device_-units * sensitivity / 2^30. Typically it works out to be the maximum g_value * 2^15.

4.11.2.40 void inv_set_accel_sample_rate (long *sample_rate_us*)

Set Accel Sample rate in micro seconds.

Parameters:

sample_rate_us Set Accel Sample rate in us

4.11.2.41 void inv_set_compass_bandwidth (int *bandwidth_hz*)

Set Compass Bandwidth in Hz.

Parameters:

bandwidth_hz Gyro bandwidth in Hz

4.11.2.42 void inv_set_compass_disturbance (int *dist*)

Set the state of a compass disturbance.

Parameters:

dist 1=disturbance, 0=no disturbance

4.11.2.43 void inv_set_compass_orientation_and_scale (int *orientation*, long *sensitivity*)

Sets the Orientation and Sensitivity of the gyro data.

Parameters:

orientation A scalar defining the transformation from chip mounting to the body frame. The function inv_orientation_matrix_to_scalar() can convert the transformation matrix to this scalar and describes the scalar in further detail.

sensitivity A scale factor to convert device units to uT such that uT = device_units ∗ sensitivity / 2^30. Typically it works out to be the maximum uT_value ∗ 2^15.

4.11.2.44 void inv_set_compass_sample_rate (long *sample_rate_us*)

Set Compass Sample rate in micro seconds.

Parameters:

sample_rate_us Set Gyro Sample rate in micro seconds.

4.11.2.45 void inv_set_compass_soft_iron_input_data (const long ∗ *data*)

This subroutine sets the compass raw data for the soft iron transformation.

Parameters:

int] the pointer of the 3x1 vector compass raw data in MPL format

4.11.2.46 void inv_set_compass_soft_iron_matrix_d (long * *matrix*)

Sets the 3x3 compass transform matrix in 32 bit Q30 fixed point format.

Parameters:

the pointer of the 3x3 matrix in Q30 format

4.11.2.47 void inv_set_compass_soft_iron_matrix_f (float * *matrix*)

Sets the 3x3 compass transform matrix in 32 bit floating point format.

Parameters:

the pointer of the 3x3 matrix in floating point format

4.11.2.48 void inv_set_gyro_bandwidth (int *bandwidth_hz*)

Set Gyro Bandwidth in Hz.

Parameters:

bandwidth_hz Gyro bandwidth in Hz

4.11.2.49 void inv_set_gyro_bias (const long * *bias*, int *accuracy*)

Sets the gyro bias.

Parameters:

bias Gyro bias in hardware units scaled by 2^16. In chip mounting frame. Length 3.

accuracy Accuracy of bias. 0 = least accurate, 3 = most accurate.

4.11.2.50 void inv_set_gyro_orientation_and_scale (int *orientation*, long *sensitivity*)

Sets the Orientation and Sensitivity of the gyro data.

Parameters:

orientation A scalar defining the transformation from chip mounting to the body frame. The function inv_orientation_matrix_to_scalar() can convert the transformation matrix to this scalar and describes the scalar in further detail.

sensitivity A scale factor to convert device units to degrees per second scaled by 2^16 such that degrees_per_second = device_units * sensitivity / 2^30. Typically it works out to be the maximum rate * 2^15.

4.11.2.51 void inv_set_gyro_sample_rate (long *sample_rate_us*)

Set Gyro Sample rate in micro seconds.

Parameters:

sample_rate_us Set Gyro Sample rate in us

4.11.2.52 void inv_set_quat_sample_rate (long *sample_rate_us*)

Set Quat Sample rate in micro seconds.

Parameters:

sample_rate_us Set Quat Sample rate in us

4.11.2.53 void inv_temperature_was_turned_off ()

This should be called when the temperature sensor has been turned off.

This is so that we will know if the data is contiguous.

4.11.2.54 void set_sensor_orientation_and_scale (struct inv_single_sensor_t * *sensor*, int *orientation*, long *sensitivity*)

Sets orientation and sensitivity field for a sensor.

Parameters:

sensor Structure to apply settings to

orientation Orientation description of how part is mounted.

sensitivity A Scale factor to convert from hardware units to standard units (dps, uT, g).

4.12 ml_math_func

Motion Library - Math Functions Common math functions the Motion Library.

Files

- file ml_math_func.c

  Math Functions.

Functions

- float inv_angle_diff (float ang1, float ang2)

  Finds the minimum angle difference ang1-ang2 such that difference is between [-M_PI,M_PI].

- short inv_big8_to_int16 (const unsigned char *big8)

  Converts a big endian byte stream into a 16-bit integer (short).

- long inv_big8_to_int32 (const unsigned char *big8)

  Converts a big endian byte stream into a 32-bit long.

- uint32_t inv_checksum (const unsigned char *str, int len)

  bernstein hash, derived from public domain source

- void inv_convert_to_body (unsigned short orientation, const long *input, long *output)

  Uses the scalar orientation value to convert from chip frame to body frame.

- void inv_convert_to_body_with_scale (unsigned short orientation, long sensitivity, const long *input, long *output)

  Uses the scalar orientation value to convert from chip frame to body frame and apply appropriate scaling.

- void inv_convert_to_chip (unsigned short orientation, const long *input, long *output)

  Uses the scalar orientation value to convert from body frame to chip frame.

- unsigned long inv_get_gyro_sum_of_sqr (const long *gyro)

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

- unsigned char * inv_int16_to_big8 (short x, unsigned char *big8)
  Converts a 16-bit short to a big endian byte stream.
- unsigned char * inv_int32_to_big8 (long x, unsigned char *big8)
  Converts a 32-bit long to a big endian byte stream.
- short inv_little8_to_int16 (const unsigned char *little8)
  Converts a little endian byte stream into a 16-bit integer (short).
- unsigned short inv_orientation_matrix_to_scalar (const signed char *mtx)
  Converts an orientation matrix made up of 0,+1,and -1 to a scalar representation.
- long inv_q29_mult (long a, long b)
  Performs a multiply and shift by 29.
- long inv_q30_mult (long a, long b)
  Performs a multiply and shift by 30.
- void inv_q_add (long *q1, long *q2, long *qSum)
  Performs a fixed point quaternion addition.
- void inv_q_mult (const long *q1, const long *q2, long *qProd)
  Performs a fixed point quaternion multiply.
- void inv_q_norm4 (float *q)
  Performs a length 4 vector normalization with a square root.
- void inv_q_rotate (const long *q, const long *in, long *out)
  Rotates a 3-element vector by Rotation defined by Q.
- long inv_q_shift_mult (long a, long b, int shift)
  Performs a multiply and shift by shift.
- void inv_quaternion_to_rotation (const long *quat, long *rot)
  Converts a quaternion to a rotation matrix.
- void inv_quaternion_to_rotation_vector (const long *quat, long *rot)
  Converts a quaternion to a rotation vector.
- double inv_vector_norm (const float *x)
  find a norm for a vector

- float inv_wrap_angle (float ang)

  Wraps angle from (-M_PI,M_PI].

4.12.1 Detailed Description

Motion Library - Math Functions Common math functions the Motion Library.

4.12.2 Function Documentation

4.12.2.1 float inv_angle_diff (float *ang1*, float *ang2*)

Finds the minimum angle difference ang1-ang2 such that difference is between [-M_PI,M_PI].

Parameters:

ang1

ang2

Returns:

angle difference ang1-ang2

4.12.2.2 void inv_convert_to_body (unsigned short *orientation*, const long * *input*, long * *output*)

Uses the scalar orientation value to convert from chip frame to body frame.

Parameters:

orientation A scalar that represent how to go from chip to body frame

input Input vector, length 3

output Output vector, length 3

4.12.2.3 void inv_convert_to_body_with_scale (unsigned short *orientation*, long *sensitivity*, const long * *input*, long * *output*)

Uses the scalar orientation value to convert from chip frame to body frame and apply appropriate scaling.

Parameters:

orientation A scalar that represent how to go from chip to body frame

sensitivity Sensitivity scale

input Input vector, length 3

output Output vector, length 3

4.12.2.4 void inv_convert_to_chip (unsigned short *orientation*, const long * *input*, long * *output*)

Uses the scalar orientation value to convert from body frame to chip frame.

Parameters:

orientation A scalar that represent how to go from chip to body frame

input Input vector, length 3

output Output vector, length 3

4.12.2.5 unsigned long inv_get_gyro_sum_of_sqr (const long * *gyro*)

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

Parameters:

gyro Gyro data scaled with 1 dps = 2^16

Returns:

the computed magnitude squared output of the gyroscope.

4.12.2.6 unsigned short inv_orientation_matrix_to_scalar (const signed char * *mtx*)

Converts an orientation matrix made up of 0,+1,and -1 to a scalar representation.

Parameters:

mtx Orientation matrix to convert to a scalar.

Returns:

Description of orientation matrix. The lowest 2 bits (0 and 1) represent the column the one is on for the first row, with the bit number 2 being the sign. The next 2 bits (3 and 4) represent the column the one is on for the second row with bit number 5 being the sign. The next 2 bits (6 and 7) represent the column the one is on for the third row with bit number 8 being the sign. In binary the identity matrix would therefor be: 010_001_000 or 0x88 in hex.

4.12.2.7 long inv_q29_mult (long *a*, long *b*)

Performs a multiply and shift by 29.

These are good functions to write in assembly on with devices with small memory where you want to get rid of the long long which some assemblers don't handle well

Parameters:

a

b

Returns:

((long long)a∗b)>>29

4.12.2.8 long inv_q30_mult (long *a*, long *b*)

Performs a multiply and shift by 30.

These are good functions to write in assembly on with devices with small memory where you want to get rid of the long long which some assemblers don't handle well

Parameters:

a

b

Returns:

((long long)a∗b)>>30

4.12.2.9 void inv_q_add (long ∗ *q1*, long ∗ *q2*, long ∗ *qSum*)

Performs a fixed point quaternion addition.

Parameters:

q1 First Quaternion term, length 4. 1.0 scaled to 2^30

q2 Second Quaternion term, length 4. 1.0 scaled to 2^30

qSum Sum after quaternion summation. Length 4. 1.0 scaled to 2^30.

4.12.2.10 void inv_q_mult (const long * *q1*, const long * *q2*, long * *qProd*)

Performs a fixed point quaternion multiply.

Parameters:

q1 First Quaternion Multicand, length 4. 1.0 scaled to 2^30

q2 Second Quaternion Multicand, length 4. 1.0 scaled to 2^30

qProd Product after quaternion multiply. Length 4. 1.0 scaled to 2^30.

4.12.2.11 void inv_q_norm4 (float * *q*)

Performs a length 4 vector normalization with a square root.

Parameters:

q vector to normalize. Returns [1,0,0,0] is magnitude is zero.

4.12.2.12 long inv_q_shift_mult (long *a*, long *b*, int *shift*)

Performs a multiply and shift by shift.

These are good functions to write in assembly on with devices with small memory where you want to get rid of the long long which some assemblers don't handle well

Parameters:

a First multicand

b Second multicand

shift Shift amount after multiplying

Returns:

((long long)a*b)<<shift

4.12.2.13 void inv_quaternion_to_rotation (const long * *quat*, long * *rot*)

Converts a quaternion to a rotation matrix.

Parameters:

quat 4-element quaternion in fixed point. One is 2^30.

rot Rotation matrix in fixed point. One is 2^30. The First 3 elements of the rotation matrix, represent the first row of the matrix. Rotation matrix multiplied by a 3 element column vector transform a vector from Body to World.

4.12.2.14 void inv_quaternion_to_rotation_vector (const long * *quat*, long * *rot*)

Converts a quaternion to a rotation vector.

A rotation vector is a method to represent a 4-element quaternion vector in 3-elements. To get the quaternion from the 3-elements, The last 3-elements of the quaternion will be the given rotation vector. The first element of the quaternion will be the positive value that will be required to make the magnitude of the quaternion 1.0 or 2^30 in fixed point units.

Parameters:

quat 4-element quaternion in fixed point. One is 2^30.

rot Rotation vector in fixed point. One is 2^30.

4.12.2.15 double inv_vector_norm (const float * *x*)

find a norm for a vector

Parameters:

a vector [3x1]

output the norm of the input vector

4.12.2.16 float inv_wrap_angle (float *ang*)

Wraps angle from (-M_PI,M_PI].

Parameters:

ang Angle in radians to wrap

Returns:

Wrapped angle from (-M_PI,M_PI]

4.13 message_layer

Motion Library - Message Layer Holds Low Occurance messages.

Files

- file message_layer.c

  Holds Low Occurance Messages.

Functions

- long inv_get_message_level_0 (int clear)

  Returns Message Flags for Level 0 Messages.

- void inv_set_message (long set, long clear, int level)

  Sets a message.

4.13.1 Detailed Description

Motion Library - Message Layer Holds Low Occurance messages.

4.13.2 Function Documentation

4.13.2.1 long inv_get_message_level_0 (int *clear*)

Returns Message Flags for Level 0 Messages.

Levels are to allow expansion of more messages in the future.

Parameters:

clear If set, will clear the message. Typically this will be set for one reader, so that you don't get the same message over and over.

Returns:

bit field to corresponding message.

4.13.2.2 void inv_set_message (long *set*, long *clear*, int *level*)

Sets a message.

Parameters:

set The flags to set.

clear Before setting anything this will clear these messages, which is useful for mutually exclusive messages such a motion or no motion message.

level Level of the messages. It starts at 0, and may increase in the future to allow more messages if the bit storage runs out.

4.14 mpl

Motion Library - Start Point Initializes MPL.

Files

- file mpl.c

  MPL start point.

Functions

- inv_error_t inv_get_version (char **version)

  used to get the MPL version.

- inv_error_t inv_init_mpl (void)

  Initializes the MPL.

- inv_error_t inv_start_mpl (void)

  Starts the MPL.

4.14.1 Detailed Description

Motion Library - Start Point Initializes MPL.

4.14.2 Function Documentation

4.14.2.1 inv_error_t inv_get_version (char ** *version*)

used to get the MPL version.

Parameters:

version a string where the MPL version gets stored.

Returns:

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

4.14.2.2 inv_error_t inv_init_mpl (void)

Initializes the MPL.

Should be called first and once

Returns:

Returns INV_SUCCESS if successful or an error code if not.

4.14.2.3 inv_error_t inv_start_mpl (void)

Starts the MPL.

Typically called after inv_init_mpl() or after a inv_stop_mpl() to start the MPL back up an running.

Returns:

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

4.15 results_holder

Motion Library - Results Holder Holds the data for MPL.

Files

- file results_holder.c
  Results Holder for HAL.

Functions

- inv_error_t inv_enable_results_holder ()
  Turns on storage of results.
- inv_error_t inv_generate_results (struct inv_sensor_cal_t *sensor_cal)
  Callback that gets called everytime there is new data.
- inv_error_t inv_get_6axis_quaternion (long *data)
  Returns a quaternion based only on gyro and accel.
- int inv_get_acc_state ()
  Gets the accel state set by inv_set_acc_state().
- 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 float data.
- void inv_get_compass_bias_error (long *bias_error)
  Get's compass bias error.
- int inv_get_compass_state ()
  Get's the compass state.
- inv_error_t inv_get_gravity (long *data)
  Gets gravity vector.
- inv_error_t inv_get_gyro_float (float *data)
  Returns 3-element vector of gyro float data.

- float inv_get_heading_confidence_interval (void)

  Get 9 axis 95% heading confidence interval for quaternion.

- int inv_get_large_mag_field ()

  Returns non-zero if there is a large magnetic field.

- 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_float (float *data)

  Returns 3-element vector of linear accel float data.

- void inv_get_local_field (long *data)

  Gets the local earth's magnetic field.

- void inv_get_mag_scale (long *data)

  Gets the compass sensitivity.

- int inv_get_motion_state (unsigned int *cntr)

  Returns the motion state.

- inv_error_t inv_get_quaternion (long *data)

  Returns a quaternion.

- inv_error_t inv_get_quaternion_float (float *data)

  Returns a quaternion.

- void inv_get_quaternion_set (long *data, int *accuracy, inv_time_t *timestamp)

  Returns a quaternion with accuracy and timestamp.

- int inv_got_accel_bias ()

  Sets state of if we know the accel bias.

- int inv_got_compass_bias ()

  Sets state of if we know the compass bias.

- inv_error_t inv_init_results_holder (void)

  Initializes results holder.

- void inv_set_acc_state (int state)

Sets the accel state.

- void inv_set_accel_bias_found (int state)
  Sets whether we know the accel bias.
- void inv_set_compass_bias_error (const long *bias_error)
  Set compass bias error.
- void inv_set_compass_bias_found (int state)
  Sets whether we know the compass bias.
- void inv_set_compass_state (int state)
  Sets the compass state.
- void inv_set_heading_confidence_interval (float ci)
  Set 9 axis 95% heading confidence interval for quaternion.
- void inv_set_large_mag_field (int state)
  Set to non-zero if there as a large magnetic field.
- void inv_set_local_field (const long *data)
  Sets the local earth's magnetic field.
- void inv_set_mag_scale (const long *data)
  Sets the compass sensitivity.
- void inv_set_motion_state (unsigned char state)
  Sets the motion state.
- inv_error_t inv_start_results_holder (void)
  Function to turn on this module.

4.15.1 Detailed Description

Motion Library - Results Holder Holds the data for MPL.

4.15.2 Function Documentation

4.15.2.1 inv_error_t inv_generate_results (struct inv_sensor_cal_t * *sensor_cal*)

Callback that gets called everytime there is new data.

It is registered by inv_start_results_holder().

Parameters:

sensor_cal New sensor data to process.

Returns:

Returns INV_SUCCESS if successful or an error code if not.

4.15.2.2 inv_error_t inv_get_6axis_quaternion (long * *data*)

Returns a quaternion based only on gyro and accel.

Parameters:

data 6-axis gyro and accel quaternion scaled such that 1.0 = 2^30.

Returns:

Returns INV_SUCCESS if successful or an error code if not.

4.15.2.3 int inv_get_acc_state ()

Gets the accel state set by inv_set_acc_state().

Returns:

accel state.

4.15.2.4 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

Returns:

INV_SUCCESS if successful INV_ERROR_INVALID_PARAMETER if invalid input pointer

4.15.2.5 inv_error_t inv_get_accel_float (float * *data*)

Returns 3-element vector of accelerometer float data.

Parameters:

data 3-element vector of accelerometer float data

Returns:

INV_SUCCESS if successful INV_ERROR_INVALID_PARAMETER if invalid input pointer

4.15.2.6 void inv_get_compass_bias_error (long * *bias_error*)

Get's compass bias error.

See inv_set_compass_bias_error() for setting.

Parameters:

bias_error Accuracy as to how well the compass bias is known. It is the error squared.

4.15.2.7 int inv_get_compass_state ()

Get's the compass state.

Returns:

the compass state that was set with inv_set_compass_state()

4.15.2.8 inv_error_t inv_get_gravity (long * *data*)

Gets gravity vector.

Parameters:

data gravity vector in body frame scaled such that 1.0 = 2^30.

Returns:

Returns INV_SUCCESS if successful or an error code if not.

4.15.2.9 inv_error_t inv_get_gyro_float (float * *data*)

Returns 3-element vector of gyro float data.

Parameters:

data 3-element vector of gyro float data

Returns:

INV_SUCCESS if successful INV_ERROR_INVALID_PARAMETER if invalid input pointer

4.15.2.10 float inv_get_heading_confidence_interval (void)

Get 9 axis 95% heading confidence interval for quaternion.

Returns:

Confidence interval in radians.

4.15.2.11 int inv_get_large_mag_field ()

Returns non-zero if there is a large magnetic field.

See inv_set_large_mag_field() for setting this variable.

Returns:

Returns non-zero if there is a large magnetic field.

4.15.2.12 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

Returns:

INV_SUCCESS if successful INV_ERROR_INVALID_PARAMETER if invalid input pointer

4.15.2.13 inv_error_t inv_get_linear_accel_float (float * *data*)

Returns 3-element vector of linear accel float data.

Parameters:

data 3-element vector of linear aceel float data

Returns:

INV_SUCCESS if successful INV_ERROR_INVALID_PARAMETER if invalid input pointer

4.15.2.14 void inv_get_local_field (long * *data*)

Gets the local earth's magnetic field.

Parameters:

data Local earth's magnetic field in uT scaled by 2^16. Length = 3. Y typically points north, Z typically points down in northern hemisphere and up in southern hemisphere.

4.15.2.15 void inv_get_mag_scale (long * *data*)

Gets the compass sensitivity.

Parameters:

data Length 3, sensitivity for each compass axis scaled such that 1.0 = 2^30.

4.15.2.16 int inv_get_motion_state (unsigned int * *cntr*)

Returns the motion state.

Parameters:

cntr Number of previous times a no motion event has occured in a row.

Returns:

Returns INV_SUCCESS if successful or an error code if not.

4.15.2.17 inv_error_t inv_get_quaternion (long * *data*)

Returns a quaternion.

Parameters:

data 9-axis quaternion scaled such that 1.0 = 2^30.

Returns:

Returns INV_SUCCESS if successful or an error code if not.

4.15.2.18 inv_error_t inv_get_quaternion_float (float * *data*)

Returns a quaternion.

Parameters:

data 9-axis quaternion.

Returns:

Returns INV_SUCCESS if successful or an error code if not.

4.15.2.19 void inv_get_quaternion_set (long * *data*, int * *accuracy*, inv_time_t * *timestamp*)

Returns a quaternion with accuracy and timestamp.

Parameters:

data 9-axis quaternion scaled such that 1.0 = 2^30.

accuracy Accuracy of quaternion, 0-3, where 3 is most accurate.

timestamp Timestamp of this quaternion in nanoseconds

4.15.2.20 int inv_got_accel_bias ()

Sets state of if we know the accel bias.

Returns:

return 1 if we know the accel bias, 0 if not. it is set with inv_set_accel_bias_found()

4.15.2.21 int inv_got_compass_bias ()

Sets state of if we know the compass bias.

Returns:

return 1 if we know the compass bias, 0 if not. it is set with inv_set_compass_bias_found()

4.15.2.22 inv_error_t inv_init_results_holder (void)

Initializes results holder.

This is called automatically by the enable function inv_enable_results_holder(). It may be called any time the feature is enabled, but is typically not needed to be called by outside callers.

Returns:

Returns INV_SUCCESS if successful or an error code if not.

4.15.2.23 void inv_set_acc_state (int *state*)

Sets the accel state.

See inv_get_acc_state() to get the value.

Parameters:

state value to set accel state to.

4.15.2.24 void inv_set_accel_bias_found (int *state*)

Sets whether we know the accel bias.

Parameters:

state Set to 1 if we know the accel bias. Can be retrieved with inv_got_accel_bias()

4.15.2.25 void inv_set_compass_bias_error (const long * *bias_error*)

Set compass bias error.

See inv_get_compass_bias_error()

Parameters:

bias_error Set's how accurate we know the compass bias. It is the error squared.

4.15.2.26 void inv_set_compass_bias_found (int *state*)

Sets whether we know the compass bias.

Parameters:

state Set to 1 if we know the compass bias. Can be retrieved with inv_got_compass_bias()

4.15.2.27 void inv_set_compass_state (int *state*)

Sets the compass state.

Parameters:

state Compass state. It can be retrieved with inv_get_compass_state().

4.15.2.28 void inv_set_heading_confidence_interval (float *ci*)

Set 9 axis 95% heading confidence interval for quaternion.

Parameters:

ci Confidence interval in radians.

4.15.2.29 void inv_set_large_mag_field (int *state*)

Set to non-zero if there as a large magnetic field.

See inv_get_large_mag_field() for getting this variable.

Parameters:

state value to set for magnetic field strength. Should be non-zero if it is large.

4.15.2.30 void inv_set_local_field (const long * *data*)

Sets the local earth's magnetic field.

Parameters:

data Local earth's magnetic field in uT scaled by 2^16. Length = 3. Y typically points north, Z typically points down in northern hemisphere and up in southern hemisphere.

4.15.2.31 void inv_set_mag_scale (const long * *data*)

Sets the compass sensitivity.

Parameters:

data Length 3, sensitivity for each compass axis scaled such that 1.0 = 2^30.

4.15.2.32 void inv_set_motion_state (unsigned char *state*)

Sets the motion state.

Parameters:

state motion state where INV_NO_MOTION is not moving and INV_MOTION is moving.

4.15.2.33 inv_error_t inv_start_results_holder (void)

Function to turn on this module.

This is automatically called by inv_enable_results_holder(). Typically not called by users.

Returns:

Returns INV_SUCCESS if successful or an error code if not.

4.16 start_manager

Motion Library - Start Manager Start Manager.

Files

- file start_manager.c

  This handles all the callbacks when inv_start_mpl() is called.

Functions

- inv_error_t inv_execute_mpl_start_notification (void)

  Callback all the functions that want to be notified when inv_start_mpl() was called.

- inv_error_t inv_init_start_manager (void)

  Initilize the start manager.

- inv_error_t inv_register_mpl_start_notification (inv_error_t(∗start_cb)(void))

  Register a callback to receive when inv_start_mpl() is called.

- inv_error_t inv_unregister_mpl_start_notification (inv_error_t(∗start_cb)(void))

  Removes a callback from start notification.

4.16.1 Detailed Description

Motion Library - Start Manager Start Manager.

4.16.2 Function Documentation

4.16.2.1 inv_error_t inv_execute_mpl_start_notification (void)

Callback all the functions that want to be notified when inv_start_mpl() was called.

Returns:

Returns INV_SUCCESS if successful or an error code if not.

4.16.2.2 inv_error_t inv_init_start_manager (void)

Initilize the start manager.

Typically called by inv_start_mpl();

Returns:

Returns INV_SUCCESS if successful or an error code if not.

4.16.2.3 inv_error_t inv_register_mpl_start_notification (inv_error_t(∗)(void) *start_cb*)

Register a callback to receive when inv_start_mpl() is called.

Parameters:

start_cb Function callback that will be called when inv_start_mpl() is called.

Returns:

Returns INV_SUCCESS if successful or an error code if not.

4.16.2.4 inv_error_t inv_unregister_mpl_start_notification (inv_error_t(∗)(void) *start_cb*)

Removes a callback from start notification.

Parameters:

start_cb function to remove from start notification

Returns:

Returns INV_SUCCESS if successful or an error code if not.

4.17 storage_manager

Motion Library - Stores Data for functions.

Files

- file storage_manager.c

  Load and Store Manager.

Defines

- #define NUM_STORAGE_BOXES 20

  Max number of entites that can be stored.

Functions

- inv_error_t inv_get_mpl_state_size (size_t *size)

  Returns the memory size needed to perform a store.

- void inv_init_storage_manager ()

  Should be called once before using any of the storage methods.

- inv_error_t inv_load_mpl_states (const unsigned char *data, size_t length)

  This function takes a block of data that has been saved in non-volatile memory and pushes to the proper locations.

- inv_error_t inv_register_load_store (inv_error_t(*load_func)(const unsigned char *data), inv_error_t(*save_func)(unsigned char *data), size_t size, unsigned int key)

  Used to register your mechanism to load and store non-volative data.

- inv_error_t inv_save_mpl_states (unsigned char *data, size_t sz)

  This function fills up a block of memory to be stored in non-volatile memory.

4.17.1 Detailed Description

Motion Library - Stores Data for functions.

4.17.2 Function Documentation

4.17.2.1 inv_error_t inv_get_mpl_state_size (size_t * *size*)

Returns the memory size needed to perform a store.

Parameters:

size Size in bytes of memory needed to store.

Returns:

Returns INV_SUCCESS if successful or an error code if not.

4.17.2.2 void inv_init_storage_manager ()

Should be called once before using any of the storage methods.

Typically called first by inv_init_mpl().

4.17.2.3 inv_error_t inv_load_mpl_states (const unsigned char * *data*, size_t *length*)

This function takes a block of data that has been saved in non-volatile memory and pushes to the proper locations.

Multiple error checks are performed on the data.

Parameters:

data Data that was saved to be loaded up by MPL

length Length of data vector in bytes

Returns:

Returns INV_SUCCESS if successful or an error code if not.

4.17.2.4 inv_error_t inv_register_load_store (inv_error_t(*)(const unsigned char *data) *load_func*, inv_error_t(*)(unsigned char *data) *save_func*, size_t *size*, unsigned int *key*)

Used to register your mechanism to load and store non-volative data.

This should typical be called during the enable function for your feature.

Parameters:

load_func function pointer you will use to receive data that was stored for you.

save_func function pointer you will use to save any data you want saved to non-volatile memory between runs.

size The size in bytes of the amount of data you want loaded and saved.

key The key associated with your data type should be unique across MPL. The key should change when your type of data for storage changes.

Returns:

Returns INV_SUCCESS if successful or an error code if not.

4.17.2.5 inv_error_t inv_save_mpl_states (unsigned char * *data*, size_t *sz*)

This function fills up a block of memory to be stored in non-volatile memory.

Parameters:

data Place to store data, size of sz, must be at least size returned by inv_get_mpl_state_size()

sz Size of data.

Returns:

Returns INV_SUCCESS if successful or an error code if not.

4.18 hal_outputs

Motion Library - HAL Outputs Sets up common outputs for HAL.

Files

- file eMPL_outputs.c

  Embedded MPL outputs.

Functions

- inv_error_t inv_disable_eMPL_outputs (void)

  Turns off creation and storage of HAL type results.

- inv_error_t inv_enable_eMPL_outputs (void)

  Turns on creation and storage of HAL type results.

- int inv_get_sensor_type_accel (long *data, int8_t *accuracy, inv_time_t *timestamp)

  Acceleration (g's) in body frame.

- int inv_get_sensor_type_compass (long *data, int8_t *accuracy, inv_time_t *timestamp)

  Magnetic field strength in body frame.

- int inv_get_sensor_type_euler (long *data, int8_t *accuracy, inv_time_t *timestamp)

  Body-to-world frame euler angles.

- int inv_get_sensor_type_gyro (long *data, int8_t *accuracy, inv_time_t *timestamp)

  Angular velocity (degrees per second) in body frame.

- int inv_get_sensor_type_heading (long *data, int8_t *accuracy, inv_time_t *timestamp)

  Quaternion-derived heading.

- int inv_get_sensor_type_quat (long *data, int8_t *accuracy, inv_time_t *timestamp)

  Body-to-world frame quaternion.

- int inv_get_sensor_type_rot_mat (long *data, int8_t *accuracy, inv_time_t *timestamp)

  Body-to-world frame rotation matrix.

4.18.1 Detailed Description

Motion Library - HAL Outputs Sets up common outputs for HAL.

4.18.2 Function Documentation

4.18.2.1 int inv_get_sensor_type_accel (long * *data*, int8_t * *accuracy*, inv_time_t * *timestamp*)

Acceleration (g's) in body frame.

Embedded MPL defines gravity as positive acceleration pointing away from the Earth.

Parameters:

data Acceleration in g's, q16 fixed point.

accuracy Accuracy of the measurement from 0 (least accurate) to 3 (most accurate).

timestamp The time in milliseconds when this sensor was read.

Returns:

1 if data was updated.

4.18.2.2 int inv_get_sensor_type_compass (long * *data*, int8_t * *accuracy*, inv_time_t * *timestamp*)

Magnetic field strength in body frame.

Parameters:

data Field strength in microteslas, q16 fixed point.

accuracy Accuracy of the measurement from 0 (least accurate) to 3 (most accurate).

timestamp The time in milliseconds when this sensor was read.

Returns:

1 if data was updated.

4.18.2.3 int inv_get_sensor_type_euler (long * *data*, int8_t * *accuracy*, inv_time_t * *timestamp*)

Body-to-world frame euler angles.

The euler angles are output with the following convention: Pitch: -180 to 180 Roll: -90 to 90 Yaw: -180 to 180

Parameters:

data Euler angles in degrees, q16 fixed point.

accuracy Accuracy of the measurement from 0 (least accurate) to 3 (most accurate).

timestamp The time in milliseconds when this sensor was read.

Returns:

1 if data was updated.

4.18.2.4 int inv_get_sensor_type_gyro (long * *data*, int8_t * *accuracy*, inv_time_t * *timestamp*)

Angular velocity (degrees per second) in body frame.

Parameters:

data Angular velocity in dps, q16 fixed point.

accuracy Accuracy of the measurement from 0 (least accurate) to 3 (most accurate).

timestamp The time in milliseconds when this sensor was read.

Returns:

1 if data was updated.

4.18.2.5 int inv_get_sensor_type_heading (long * *data*, int8_t * *accuracy*, inv_time_t * *timestamp*)

Quaternion-derived heading.

Parameters:

data Heading in degrees, q16 fixed point.

accuracy Accuracy of the measurement from 0 (least accurate) to 3 (most accurate).

timestamp The time in milliseconds when this sensor was read.

Returns:

1 if data was updated.

4.18.2.6 int inv_get_sensor_type_quat (long * *data*, int8_t * *accuracy*, inv_time_t * *timestamp*)

Body-to-world frame quaternion.

The elements are output in the following order: W, X, Y, Z.

Parameters:

data Quaternion, q30 fixed point.

accuracy Accuracy of the measurement from 0 (least accurate) to 3 (most accurate).

timestamp The time in milliseconds when this sensor was read.

Returns:

1 if data was updated.

4.18.2.7 int inv_get_sensor_type_rot_mat (long * *data*, int8_t * *accuracy*, inv_time_t * *timestamp*)

Body-to-world frame rotation matrix.

Parameters:

data Rotation matrix, q30 fixed point.

accuracy Accuracy of the measurement from 0 (least accurate) to 3 (most accurate).

timestamp The time in milliseconds when this sensor was read.

Returns:

1 if data was updated.

4.19 MSP430 System Layer

MSP430 System Layer APIs.

Files

- file msp430_clock.h

  Functions to configure the MSP430 system clock to settings required for eMPL.

- file msp430_i2c.h

  Serial communication functions needed by eMPL to communicate to the MPU devices.

- file msp430_interrupt.h

  Supports common interrupt vectors using callbacks.

- file packet.h

  Defines needed for sending data/debug packets via USB.

Functions

- void eMPL_send_data (unsigned char type, long *data)

  Send a data packet via USB.

- void eMPL_send_quat (long *quat)

  Send a quaternion packet via USB.

- int msp430_clock_disable (void)

  Disable the millisecond timer.

- int msp430_clock_enable (void)

  Enable the millisecond timer.

- int msp430_clock_init (unsigned long mclk, unsigned char xt)

  Set the frequency of MCLK, SMCLK, and ACLK.

- int msp430_delay_ms (unsigned long num_ms)

  Perform a blocking delay.

- int msp430_get_clock_ms (unsigned long *count)

  Get current clock count.

- int msp430_get_smclk_freq (unsigned long *smclk)
  Get frequency of SMCLK.
- int msp430_i2c_disable (void)
  Disable I2C communication.
- int msp430_i2c_enable (void)
  Set up the I2C port and configure the MSP430 as the master.
- int msp430_i2c_read (unsigned char slave_addr, unsigned char reg_addr, unsigned char length, unsigned char *data)
  Read from a device.
- int msp430_i2c_write (unsigned char slave_addr, unsigned char reg_addr, unsigned char length, unsigned char const *data)
  Write to a device register.
- int msp430_int_disable (void)
  Disable interrupts.
- int msp430_int_enable (void)
  Enable interrupts.
- int msp430_int_init (void)
  Set up shared interrupt vectors.
- int msp430_reg_int_cb (void(*cb)(void), unsigned short pin, unsigned char lp_exit, unsigned char active_low)
  Register callback for a particular interrupt pin.
- int msp430_register_timer_cb (void(*timer_cb)(void), unsigned long num_ms)
  Register a one-time timer event.
- int msp430_slow_timer (unsigned char slow)
  Slow down the timer.

4.19.1 Detailed Description

MSP430 System Layer APIs.

To interface with any platform, eMPL needs access to various system layer functions.

4.19.2 Function Documentation

4.19.2.1 void eMPL_send_data (unsigned char *type*, long * *data*)

Send a data packet via USB.

Parameters:

type Contents of packet (PACKET_DATA_ACCEL, etc).

data Data (length dependent on contents).

4.19.2.2 void eMPL_send_quat (long * *quat*)

Send a quaternion packet via USB.

The host is expected to use the data in this packet to graphically represent the device orientation. To send quaternion in the same manner as any other data packet, use eMPL_send_data.

Parameters:

quat Quaternion data.

4.19.2.3 int msp430_clock_disable (void)

Disable the millisecond timer.

This function should be used prior to entering a low-power mode.

Returns:

0 if successful.

4.19.2.4 int msp430_clock_enable (void)

Enable the millisecond timer.

This function is automatically called by *msp430_clock_init*. It should be used to re-enable the timer after *msp430_clock_disable* is called.

Returns:

0 if successful.

4.19.2.5 int msp430_clock_init (unsigned long *mclk*, unsigned char *xt*)

Set the frequency of MCLK, SMCLK, and ACLK.

Parameters:

mclk Frequency of master clock.

xt 1 if XT1 is present, 2 if XT2 is present, 0 otherwise.

Returns:

0 if successful.

4.19.2.6 int msp430_delay_ms (unsigned long *num_ms*)

Perform a blocking delay.

Parameters:

num_ms Number of milliseconds to delay.

Returns:

0 if successful.

4.19.2.7 int msp430_get_clock_ms (unsigned long * *count*)

Get current clock count.

Timer overflow will occur after 2^32 milliseconds.

Parameters:

count Timer count in milliseconds.

Returns:

0 if successful.

4.19.2.8 int msp430_get_smclk_freq (unsigned long * *smclk*)

Get frequency of SMCLK.

Currently, the sub-master clock and the master clock are the same frequency.

Parameters:

smclk SMCLK frequency.

Returns:

0 if successful.

4.19.2.9 int msp430_i2c_disable (void)

Disable I2C communication.

This function will disable the I2C hardware and should be called prior to entering low-power mode.

Returns:

0 if successful.

4.19.2.10 int msp430_i2c_enable (void)

Set up the I2C port and configure the MSP430 as the master.

Returns:

0 if successful.

4.19.2.11 int msp430_i2c_read (unsigned char *slave_addr*, unsigned char *reg_addr*, unsigned char *length*, unsigned char * *data*)

Read from a device.

Parameters:

slave_addr Slave address of device.

reg_addr Slave register to be read from.

length Number of bytes to read.

data Data from register.

Returns:

0 if successful.

4.19.2.12 int msp430_i2c_write (unsigned char *slave_addr*, unsigned char *reg_addr*, unsigned char *length*, unsigned char const * *data*)

Write to a device register.

Parameters:

slave_addr Slave address of device.

reg_addr Slave register to be written to.

length Number of bytes to write.

data Data to be written to register.

Returns:

0 if successful.

4.19.2.13 int msp430_int_disable (void)

Disable interrupts.

Returns:

0 if successful.

4.19.2.14 int msp430_int_enable (void)

Enable interrupts.

Returns:

0 if successful.

4.19.2.15 int msp430_int_init (void)

Set up shared interrupt vectors.

This function will automatically call *msp430_int_enable* before returning.

Returns:

0 if successful.

4.19.2.16 int msp430_reg_int_cb (void(∗)(void) *cb*, unsigned short *pin*, unsigned char *lp_exit*, unsigned char *active_low*)

Register callback for a particular interrupt pin.

This function will override any function already registered.

If *cb* is set to NULL, this interrupt will be disabled.

Parameters:

cb Function executed for this interrupt.

pin Port/pin number (INT_PIN_Pxx).

lp_exit Low-power mode exited after this interrupt (INT_EXIT_LPMx).

active_low 1 if this interrupt is active low.

Returns:

0 if successful.

4.19.2.17 int msp430_register_timer_cb (void(∗)(void) *timer_cb*, unsigned long *num_ms*)

Register a one-time timer event.

Only one event can be registered. If this function is called before the current timer ends, the new event will be registered and the current one will be discarded.

Parameters:

timer_cb Function called when timer is expired.

num_ms Number of milliseconds before function is called.

Returns:

0 if successful.

4.19.2.18 int msp430_slow_timer (unsigned char *slow*)

Slow down the timer.

By default, a millisecond timer is used for timing/scheduling purposes. This API can be used to slow down the interval at which this clock is updated, saving power by reducing interrupts.

Parameters:

slow 1 to slow down timer.

Returns:

0 if successful.

4.20 Sensor Driver Layer

Hardware drivers to communicate with sensors via I2C.

Files

- file inv_mpu.c

  An I2C-based driver for Invensense gyroscopes.

- file inv_mpu_dmp_motion_driver.c

  DMP image and interface functions.

Functions

- int dmp_enable_6x_lp_quat (unsigned char enable)

  Generate 6-axis quaternions from the DMP.

- int dmp_enable_feature (unsigned char mask)

  Enable DMP features.

- int dmp_enable_lp_quat (unsigned char enable)

  Generate 3-axis quaternions from the DMP.

- int dmp_get_fifo_rate (unsigned short *rate)

  Get DMP output rate.

- int dmp_get_pedometer_step_count (unsigned long *count)

  Get current step count.

- int dmp_get_pedometer_walk_time (unsigned long *time)

  Get duration of walking time.

- int dmp_load_motion_driver_firmware (void)

  Load the DMP with this image.

- int dmp_read_fifo (short *gyro, short *accel, long *quat, unsigned long *timestamp, short *sensors, unsigned char *more)

  Get one packet from the FIFO.

- int dmp_register_orient_cb (void(*func)(unsigned char))

Register a function to be executed on an orientation event.

- int dmp_register_tap_cb (void(∗func)(unsigned char, unsigned char))
  Register a function to be executed on a tap event.
- int dmp_set_fifo_rate (unsigned short rate)
  Set DMP output rate.
- int dmp_set_gyro_bias (long ∗bias)
  Push gyro biases to the DMP.
- int dmp_set_interrupt_mode (unsigned char mode)
  Specify when a DMP interrupt should occur.
- int dmp_set_orient_axes (unsigned char axis)
  Set which orientations will trigger an event.
- int dmp_set_orient_thresh (float angle)
  Set orientation angle threshold.
- int dmp_set_orient_time (unsigned short time)
  Set orientation time.
- int dmp_set_orientation (unsigned short orient)
  Push gyro and accel orientation to the DMP.
- int dmp_set_pedometer_step_count (unsigned long count)
  Overwrite current step count.
- int dmp_set_pedometer_walk_time (unsigned long time)
  Overwrite current walk time.
- int dmp_set_shake_reject_thresh (long sf, unsigned short thresh)
  Set shake rejection threshold.
- int dmp_set_shake_reject_time (unsigned short time)
  Set shake rejection time.
- int dmp_set_shake_reject_timeout (unsigned short time)
  Set shake rejection timeout.
- int dmp_set_tap_axes (unsigned char axis)

Set which axes will register a tap.

- int dmp_set_tap_count (unsigned char min_taps)
  Set minimum number of taps needed for an interrupt.
- int dmp_set_tap_thresh (unsigned char axis, unsigned short thresh)
  Set tap threshold for a specific axis.
- int dmp_set_tap_time (unsigned short time)
  Set length between valid taps.
- int dmp_set_tap_time_multi (unsigned short time)
  Set max time between taps to register as a multi-tap.
- int mpu_configure_fifo (unsigned char sensors)
  Select which sensors are pushed to FIFO.
- int mpu_get_accel_fsr (unsigned char *fsr)
  Get the accel full-scale range.
- int mpu_get_accel_reg (short *data, unsigned long *timestamp)
  Read raw accel data directly from the registers.
- int mpu_get_accel_sens (unsigned short *sens)
  Get accel sensitivity scale factor.
- int mpu_get_compass_fsr (unsigned short *fsr)
  Get the compass full-scale range.
- int mpu_get_compass_reg (short *data, unsigned long *timestamp)
  Read raw compass data.
- int mpu_get_compass_sample_rate (unsigned short *rate)
  Get compass sampling rate.
- int mpu_get_dmp_state (unsigned char *enabled)
  Get DMP state.
- int mpu_get_fifo_config (unsigned char *sensors)
  Get current FIFO configuration.
- int mpu_get_gyro_fsr (unsigned short *fsr)

Get the gyro full-scale range.

- int mpu_get_gyro_reg (short *data, unsigned long *timestamp)

  Read raw gyro data directly from the registers.

- int mpu_get_gyro_sens (float *sens)

  Get gyro sensitivity scale factor.

- int mpu_get_lpf (unsigned short *lpf)

  Get the current DLPF setting.

- int mpu_get_power_state (unsigned char *power_on)

  Get current power state.

- int mpu_get_sample_rate (unsigned short *rate)

  Get sampling rate.

- int mpu_get_temperature (long *data, unsigned long *timestamp)

  Read temperature data directly from the registers.

- int mpu_init (struct int_param_s *int_param)

  Initialize hardware.

- int mpu_load_firmware (unsigned short length, const unsigned char *firmware, unsigned short start_addr, unsigned short sample_rate)

  Load and verify DMP image.

- int mpu_lp_accel_mode (unsigned char rate)

  Enter low-power accel-only mode.

- int mpu_lp_motion_interrupt (unsigned short thresh, unsigned char time, unsigned char lpa_freq)

  Enters LP accel motion interrupt mode.

- int mpu_read_fifo (short *gyro, short *accel, unsigned long *timestamp, unsigned char *sensors, unsigned char *more)

  Get one packet from the FIFO.

- int mpu_read_fifo_stream (unsigned short length, unsigned char *data, unsigned char *more)

  Get one unparsed packet from the FIFO.

- int mpu_read_mem (unsigned short mem_addr, unsigned short length, unsigned char *data)

  Read from the DMP memory.

- int mpu_read_reg (unsigned char reg, unsigned char *data)

  Read from a single register.

- int mpu_reg_dump (void)

  Register dump for testing.

- int mpu_reset_fifo (void)

  Reset FIFO read/write pointers.

- int mpu_run_self_test (long *gyro, long *accel)

  Trigger gyro/accel/compass self-test.

- int mpu_set_accel_bias (const long *accel_bias)

  Push biases to the accel bias registers.

- int mpu_set_accel_fsr (unsigned char fsr)

  Set the accel full-scale range.

- int mpu_set_bypass (unsigned char bypass_on)

  Set device to bypass mode.

- int mpu_set_compass_sample_rate (unsigned short rate)

  Set compass sampling rate.

- int mpu_set_dmp_state (unsigned char enable)

  Enable/disable DMP support.

- int mpu_set_gyro_fsr (unsigned short fsr)

  Set the gyro full-scale range.

- int mpu_set_int_latched (unsigned char enable)

  Enable latched interrupts.

- int mpu_set_int_level (unsigned char active_low)

  Set interrupt level.

- int mpu_set_lpf (unsigned short lpf)

  Set digital low pass filter.

- int mpu_set_sample_rate (unsigned short rate)

  Set sampling rate.

- int mpu_set_sensors (unsigned char sensors)

  Turn specific sensors on/off.

- int mpu_write_mem (unsigned short mem_addr, unsigned short length, unsigned char ∗data)

  Write to the DMP memory.

4.20.1 Detailed Description

Hardware drivers to communicate with sensors via I2C.

4.20.2 Function Documentation

4.20.2.1 int dmp_enable_6x_lp_quat (unsigned char *enable*)

Generate 6-axis quaternions from the DMP.

In this driver, the 3-axis and 6-axis DMP quaternion features are mutually exclusive.

Parameters:

enable 1 to enable 6-axis quaternion.

Returns:

0 if successful.

4.20.2.2 int dmp_enable_feature (unsigned char *mask*)

Enable DMP features.

The following #define's are used in the input mask:

DMP_FEATURE_TAP

DMP_FEATURE_ORIENTATION

DMP_FEATURE_LP_QUAT

DMP_FEATURE_6X_LP_QUAT

NOTE: Gyro and accel data are always put into the FIFO.

NOTE: DMP_FEATURE_LP_QUAT and DMP_FEATURE_6X_LP_QUAT are mutually exclusive.

Parameters:

mask Mask of features to enable.

Returns:

0 if successful.

4.20.2.3 int dmp_enable_lp_quat (unsigned char *enable*)

Generate 3-axis quaternions from the DMP.

In this driver, the 3-axis and 6-axis DMP quaternion features are mutually exclusive.

Parameters:

enable 1 to enable 3-axis quaternion.

Returns:

0 if successful.

4.20.2.4 int dmp_get_fifo_rate (unsigned short * *rate*)

Get DMP output rate.

Parameters:

rate Current fifo rate (Hz).

Returns:

0 if successful.

4.20.2.5 int dmp_get_pedometer_step_count (unsigned long * *count*)

Get current step count.

Parameters:

count Number of steps detected.

Returns:

0 if successful.

4.20.2.6 int dmp_get_pedometer_walk_time (unsigned long * *time*)

Get duration of walking time.

Parameters:

time Walk time in milliseconds.

Returns:

0 if successful.

4.20.2.7 int dmp_load_motion_driver_firmware (void)

Load the DMP with this image.

Returns:

0 if successful.

4.20.2.8 int dmp_read_fifo (short * *gyro*, short * *accel*, long * *quat*, unsigned long * *timestamp*, short * *sensors*, unsigned char * *more*)

Get one packet from the FIFO.

If *sensors* does not contain a particular sensor, disregard the data returned to that pointer.

sensors can contain a combination of the following flags:

INV_X_GYRO, INV_Y_GYRO, INV_Z_GYRO

INV_XYZ_GYRO

INV_XYZ_ACCEL

INV_WXYZ_QUAT

If the FIFO has no new data, *sensors* will be zero.

If the FIFO is disabled, *sensors* will be zero and this function will return a non-zero error code.

Parameters:

gyro Gyro data in hardware units.
accel Accel data in hardware units.
quat 3-axis quaternion data in hardware units.
timestamp Timestamp in milliseconds.
sensors Mask of sensors read from FIFO.
more Number of remaining packets.

Returns:

0 if successful.

4.20.2.9 int dmp_register_orient_cb (void(∗)(unsigned char) *func*)

Register a function to be executed on an orientation event.

Parameters:

func Callback function.

Returns:

0 if successful.

4.20.2.10 int dmp_register_tap_cb (void(∗)(unsigned char, unsigned char) *func*)

Register a function to be executed on a tap event.

The tap direction is represented by one of the following:

TAP_X_UP

TAP_X_DOWN

TAP_Y_UP

TAP_Y_DOWN

TAP_Z_UP

TAP_Z_DOWN

Parameters:

func Callback function.

Returns:

0 if successful.

4.20.2.11 int dmp_set_fifo_rate (unsigned short *rate*)

Set DMP output rate.

Only used when DMP is on.

Parameters:

rate Desired fifo rate (Hz).

Returns:

0 if successful.

4.20.2.12 int dmp_set_gyro_bias (long * *bias*)

Push gyro biases to the DMP.

Because the gyro integration is handled in the DMP, any gyro biases calculated by the MPL should be pushed down to DMP memory to remove 3-axis quaternion drift.

Parameters:

bias Gyro biases in q16.

Returns:

0 if successful.

4.20.2.13 int dmp_set_interrupt_mode (unsigned char *mode*)

Specify when a DMP interrupt should occur.

A DMP interrupt can be configured to trigger on either of the two conditions below:

a. One FIFO period has elapsed (set by *mpu_set_sample_rate*).

b. A tap event has been detected.

Parameters:

mode DMP_INT_GESTURE or DMP_INT_CONTINUOUS.

Returns:

0 if successful.

4.20.2.14 int dmp_set_orient_axes (unsigned char *axis*)

Set which orientations will trigger an event.

This function expects a mask containing a combination of the following macros:

ORIENTATION_X_UP

ORIENTATION_X_DOWN

ORIENTATION_Y_UP

ORIENTATION_Y_DOWN

ORIENTATION_Z_UP

ORIENTATION_Z_DOWN

ORIENTATION_FLIP

Parameters:

axis 1, 2, and 4 for XYZ, respectively.

Returns:

0 if successful.

4.20.2.15 int dmp_set_orient_thresh (float *angle*)

Set orientation angle threshold.

Parameters:

angle Angle where orientation changes.

Returns:

0 if successful.

4.20.2.16 int dmp_set_orient_time (unsigned short *time*)

Set orientation time.

Sets the length of time that the device must remain in the same orientation before a DMP orientation event will occur. A mandatory 60 ms is added to this parameter.

Parameters:

time Time in milliseconds.

Returns:

0 if successful.

4.20.2.17 int dmp_set_orientation (unsigned short *orient*)

Push gyro and accel orientation to the DMP.

The orientation is represented here as the output of *inv_orientation_matrix_to_scalar*.

Parameters:

orient Gyro and accel orientation in body frame.

Returns:

0 if successful.

4.20.2.18 int dmp_set_pedometer_step_count (unsigned long *count*)

Overwrite current step count.

WARNING: This function writes to DMP memory and could potentially encounter a race condition if called while the pedometer is enabled.

Parameters:

count New step count.

Returns:

0 if successful.

4.20.2.19 int dmp_set_pedometer_walk_time (unsigned long *time*)

Overwrite current walk time.

WARNING: This function writes to DMP memory and could potentially encounter a race condition if called while the pedometer is enabled.

Parameters:

time New walk time in milliseconds.

4.20.2.20 int dmp_set_shake_reject_thresh (long *sf*, unsigned short *thresh*)

Set shake rejection threshold.

If the DMP detects a gyro sample larger than *thresh*, taps are rejected.

Parameters:

sf Gyro scale factor.

thresh Gyro threshold in dps.

Returns:

0 if successful.

4.20.2.21 int dmp_set_shake_reject_time (unsigned short *time*)

Set shake rejection time.

Sets the length of time that the gyro must be outside of the threshold set by *gyro_set_shake_reject_thresh* before taps are rejected. A mandatory 60 ms is added to this parameter.

Parameters:

time Time in milliseconds.

Returns:

0 if successful.

4.20.2.22 int dmp_set_shake_reject_timeout (unsigned short *time*)

Set shake rejection timeout.

Sets the length of time after a shake rejection that the gyro must stay inside of the threshold before taps can be detected again. A mandatory 60 ms is added to this parameter.

Parameters:

time Time in milliseconds.

Returns:

0 if successful.

4.20.2.23 int dmp_set_tap_axes (unsigned char *axis*)

Set which axes will register a tap.

Parameters:

axis 1, 2, and 4 for XYZ, respectively.

Returns:

0 if successful.

4.20.2.24 int dmp_set_tap_count (unsigned char *min_taps*)

Set minimum number of taps needed for an interrupt.

Parameters:

min_taps Minimum consecutive taps (1-4).

Returns:

0 if successful.

4.20.2.25 int dmp_set_tap_thresh (unsigned char *axis*, unsigned short *thresh*)

Set tap threshold for a specific axis.

Parameters:

axis 1, 2, and 4 for XYZ accel, respectively.
thresh Tap threshold, in mg/ms.

Returns:

0 if successful.

4.20.2.26 int dmp_set_tap_time (unsigned short *time*)

Set length between valid taps.

Parameters:

time Milliseconds between taps.

Returns:

0 if successful.

4.20.2.27 int dmp_set_tap_time_multi (unsigned short *time*)

Set max time between taps to register as a multi-tap.

Parameters:

time Max milliseconds between taps.

Returns:

0 if successful.

4.20.2.28 int mpu_configure_fifo (unsigned char *sensors*)

Select which sensors are pushed to FIFO.

sensors can contain a combination of the following flags:

INV_X_GYRO, INV_Y_GYRO, INV_Z_GYRO

INV_XYZ_GYRO

INV_XYZ_ACCEL

Parameters:

sensors Mask of sensors to push to FIFO.

Returns:

0 if successful.

4.20.2.29 int mpu_get_accel_fsr (unsigned char * *fsr*)

Get the accel full-scale range.

Parameters:

fsr Current full-scale range.

Returns:

0 if successful.

4.20.2.30 int mpu_get_accel_reg (short * *data*, unsigned long * *timestamp*)

Read raw accel data directly from the registers.

Parameters:

data Raw data in hardware units.

timestamp Timestamp in milliseconds. Null if not needed.

Returns:

0 if successful.

4.20.2.31 int mpu_get_accel_sens (unsigned short * *sens*)

Get accel sensitivity scale factor.

Parameters:

sens Conversion from hardware units to g's.

Returns:

0 if successful.

4.20.2.32 int mpu_get_compass_fsr (unsigned short * *fsr*)

Get the compass full-scale range.

Parameters:

fsr Current full-scale range.

Returns:

0 if successful.

4.20.2.33 int mpu_get_compass_reg (short * *data*, unsigned long * *timestamp*)

Read raw compass data.

Parameters:

data Raw data in hardware units.

timestamp Timestamp in milliseconds. Null if not needed.

Returns:

0 if successful.

4.20.2.34 int mpu_get_compass_sample_rate (unsigned short * *rate*)

Get compass sampling rate.

Parameters:

rate Current compass sampling rate (Hz).

Returns:

0 if successful.

4.20.2.35 int mpu_get_dmp_state (unsigned char * *enabled*)

Get DMP state.

Parameters:

enabled 1 if enabled.

Returns:

0 if successful.

4.20.2.36 int mpu_get_fifo_config (unsigned char * *sensors*)

Get current FIFO configuration.

sensors can contain a combination of the following flags:

INV_X_GYRO, INV_Y_GYRO, INV_Z_GYRO

INV_XYZ_GYRO

INV_XYZ_ACCEL

Parameters:

sensors Mask of sensors in FIFO.

Returns:

0 if successful.

4.20.2.37 int mpu_get_gyro_fsr (unsigned short * *fsr*)

Get the gyro full-scale range.

Parameters:

fsr Current full-scale range.

Returns:

0 if successful.

4.20.2.38 int mpu_get_gyro_reg (short * *data*, unsigned long * *timestamp*)

Read raw gyro data directly from the registers.

Parameters:

data Raw data in hardware units.

timestamp Timestamp in milliseconds. Null if not needed.

Returns:

0 if successful.

4.20.2.39 int mpu_get_gyro_sens (float * *sens*)

Get gyro sensitivity scale factor.

Parameters:

sens Conversion from hardware units to dps.

Returns:

0 if successful.

4.20.2.40 int mpu_get_lpf (unsigned short * *lpf*)

Get the current DLPF setting.

Parameters:

lpf Current LPF setting. 0 if successful.

4.20.2.41 int mpu_get_power_state (unsigned char * *power_on*)

Get current power state.

Parameters:

power_on 1 if turned on, 0 if suspended.

Returns:

0 if successful.

4.20.2.42 int mpu_get_sample_rate (unsigned short * *rate*)

Get sampling rate.

Parameters:

rate Current sampling rate (Hz).

Returns:

0 if successful.

4.20.2.43 int mpu_get_temperature (long * *data*, unsigned long * *timestamp*)

Read temperature data directly from the registers.

Parameters:

data Data in q16 format.

timestamp Timestamp in milliseconds. Null if not needed.

Returns:

0 if successful.

4.20.2.44 int mpu_init (struct int_param_s * *int_param*)

Initialize hardware.

Initial configuration:

Gyro FSR: +/- 2000DPS

Accel FSR +/- 2G

DLPF: 42Hz

FIFO rate: 50Hz

Clock source: Gyro PLL

FIFO: Disabled.

Data ready interrupt: Disabled, active low, unlatched.

Parameters:

int_param Platform-specific parameters to interrupt API.

Returns:

0 if successful.

4.20.2.45 int mpu_load_firmware (unsigned short *length*, const unsigned char ** *firmware*, unsigned short *start_addr*, unsigned short *sample_rate*)

Load and verify DMP image.

Parameters:

length Length of DMP image.

firmware DMP code.

start_addr Starting address of DMP code memory.

sample_rate Fixed sampling rate used when DMP is enabled.

Returns:

0 if successful.

4.20.2.46 int mpu_lp_accel_mode (unsigned char *rate*)

Enter low-power accel-only mode.

In low-power accel mode, the chip goes to sleep and only wakes up to sample the accelerometer at one of the following frequencies:

MPU6050: 1.25Hz, 5Hz, 20Hz, 40Hz

MPU6500: 1.25Hz, 2.5Hz, 5Hz, 10Hz, 20Hz, 40Hz, 80Hz, 160Hz, 320Hz, 640Hz

If the requested rate is not one listed above, the device will be set to the next highest rate. Requesting a rate above the maximum supported frequency will result in an error.

To select a fractional wake-up frequency, round down the value passed to *rate*.

Parameters:

rate Minimum sampling rate, or zero to disable LP accel mode.

Returns:

0 if successful.

4.20.2.47 int mpu_lp_motion_interrupt (unsigned short *thresh*, unsigned char *time*, unsigned char *lpa_freq*)

Enters LP accel motion interrupt mode.

The behavior of this feature is very different between the MPU6050 and the MPU6500. Each chip's version of this feature is explained below.

MPU6050:

When this mode is first enabled, the hardware captures a single accel sample, and subsequent samples are compared with this one to determine if the device is in motion. Therefore, whenever this "locked" sample needs to be changed, this function must be called again.

The hardware motion threshold can be between 32mg and 8160mg in 32mg increments.

Low-power accel mode supports the following frequencies:

1.25Hz, 5Hz, 20Hz, 40Hz

MPU6500:

Unlike the MPU6050 version, the hardware does not "lock in" a reference sample. The hardware monitors the accel data and detects any large change over a short period of time.

The hardware motion threshold can be between 4mg and 1020mg in 4mg increments.

MPU6500 Low-power accel mode supports the following frequencies:

1.25Hz, 2.5Hz, 5Hz, 10Hz, 20Hz, 40Hz, 80Hz, 160Hz, 320Hz, 640Hz

NOTES:

The driver will round down *thresh* to the nearest supported value if an unsupported threshold is selected.

To select a fractional wake-up frequency, round down the value passed to *lpa_freq*.

The MPU6500 does not support a delay parameter. If this function is used for the MPU6500, the value passed to *time* will be ignored.

To disable this mode, set *lpa_freq* to zero. The driver will restore the previous configuration.

Parameters:

thresh Motion threshold in mg.

time Duration in milliseconds that the accel data must exceed *thresh* before motion is reported.

lpa_freq Minimum sampling rate, or zero to disable.

Returns:

0 if successful.

4.20.2.48 int mpu_read_fifo (short * *gyro*, short * *accel*, unsigned long * *timestamp*, unsigned char * *sensors*, unsigned char * *more*)

Get one packet from the FIFO.

If *sensors* does not contain a particular sensor, disregard the data returned to that pointer.

sensors can contain a combination of the following flags:

INV_X_GYRO, INV_Y_GYRO, INV_Z_GYRO

INV_XYZ_GYRO

INV_XYZ_ACCEL

If the FIFO has no new data, *sensors* will be zero.

If the FIFO is disabled, *sensors* will be zero and this function will return a non-zero error code.

Parameters:

gyro Gyro data in hardware units.

accel Accel data in hardware units.

timestamp Timestamp in milliseconds.

sensors Mask of sensors read from FIFO.

more Number of remaining packets.

Returns:

0 if successful.

4.20.2.49 int mpu_read_fifo_stream (unsigned short *length*, unsigned char * *data*, unsigned char * *more*)

Get one unparsed packet from the FIFO.

This function should be used if the packet is to be parsed elsewhere.

Parameters:

length Length of one FIFO packet.

data FIFO packet.

more Number of remaining packets.

4.20.2.50 int mpu_read_mem (unsigned short *mem_addr*, unsigned short *length*, unsigned char * *data*)

Read from the DMP memory.

This function prevents I2C reads past the bank boundaries. The DMP memory is only accessible when the chip is awake.

Parameters:

mem_addr Memory location (bank << 8 | start address)

length Number of bytes to read.

data Bytes read from memory.

Returns:

0 if successful.

4.20.2.51 int mpu_read_reg (unsigned char *reg*, unsigned char * *data*)

Read from a single register.

NOTE: The memory and FIFO read/write registers cannot be accessed.

Parameters:

reg Register address.

data Register data.

Returns:

0 if successful.

4.20.2.52 int mpu_reg_dump (void)

Register dump for testing.

Returns:

0 if successful.

4.20.2.53 int mpu_reset_fifo (void)

Reset FIFO read/write pointers.

Returns:

0 if successful.

4.20.2.54 int mpu_run_self_test (long * *gyro*, long * *accel*)

Trigger gyro/accel/compass self-test.

On success/error, the self-test returns a mask representing the sensor(s) that failed. For each bit, a one (1) represents a "pass" case; conversely, a zero (0) indicates a failure.

The mask is defined as follows:

Bit 0: Gyro.

Bit 1: Accel.

Bit 2: Compass.

Currently, the hardware self-test is unsupported for MPU6500. However, this function can still be used to obtain the accel and gyro biases.

Parameters:

gyro Gyro biases in q16 format.

accel Accel biases (if applicable) in q16 format.

Returns:

Result mask (see above).

4.20.2.55 int mpu_set_accel_bias (const long * *accel_bias*)

Push biases to the accel bias registers.

This function expects biases relative to the current sensor output, and these biases will be added to the factory-supplied values.

Parameters:

accel_bias New biases.

Returns:

0 if successful.

4.20.2.56 int mpu_set_accel_fsr (unsigned char *fsr*)

Set the accel full-scale range.

Parameters:

fsr Desired full-scale range.

Returns:

0 if successful.

4.20.2.57 int mpu_set_bypass (unsigned char *bypass_on*)

Set device to bypass mode.

Parameters:

bypass_on 1 to enable bypass mode.

Returns:

0 if successful.

4.20.2.58 int mpu_set_compass_sample_rate (unsigned short *rate*)

Set compass sampling rate.

The compass on the auxiliary I2C bus is read by the MPU hardware at a maximum of 100Hz. The actual rate can be set to a fraction of the gyro sampling rate.

WARNING: The new rate may be different than what was requested. Call mpu_get_-compass_sample_rate to check the actual setting.

Parameters:

rate Desired compass sampling rate (Hz).

Returns:

0 if successful.

4.20.2.59 int mpu_set_dmp_state (unsigned char *enable*)

Enable/disable DMP support.

Parameters:

enable 1 to turn on the DMP.

Returns:

0 if successful.

4.20.2.60 int mpu_set_gyro_fsr (unsigned short *fsr*)

Set the gyro full-scale range.

Parameters:

fsr Desired full-scale range.

Returns:

0 if successful.

4.20.2.61 int mpu_set_int_latched (unsigned char *enable*)

Enable latched interrupts.

Any MPU register will clear the interrupt.

Parameters:

enable 1 to enable, 0 to disable.

Returns:

0 if successful.

4.20.2.62 int mpu_set_int_level (unsigned char *active_low*)

Set interrupt level.

Parameters:

active_low 1 for active low, 0 for active high.

Returns:

0 if successful.

4.20.2.63 int mpu_set_lpf (unsigned short *lpf*)

Set digital low pass filter.

The following LPF settings are supported: 188, 98, 42, 20, 10, 5.

Parameters:

lpf Desired LPF setting.

Returns:

0 if successful.

4.20.2.64 int mpu_set_sample_rate (unsigned short *rate*)

Set sampling rate.

Sampling rate must be between 4Hz and 1kHz.

Parameters:

rate Desired sampling rate (Hz).

Returns:

0 if successful.

4.20.2.65 int mpu_set_sensors (unsigned char *sensors*)

Turn specific sensors on/off.

sensors can contain a combination of the following flags:

INV_X_GYRO, INV_Y_GYRO, INV_Z_GYRO

INV_XYZ_GYRO

INV_XYZ_ACCEL

INV_XYZ_COMPASS

Parameters:

sensors Mask of sensors to wake.

Returns:

0 if successful.

4.20.2.66 int mpu_write_mem (unsigned short *mem_addr*, unsigned short *length*, unsigned char * *data*)

Write to the DMP memory.

This function prevents I2C writes past the bank boundaries. The DMP memory is only accessible when the chip is awake.

Parameters:

mem_addr Memory location (bank << 8 | start address)

length Number of bytes to write.

data Bytes to write to memory.

Returns:

0 if successful.