CrossWorks Fixed Point Library

Version: 3.0

Contents

CrossWorks Fixed Point Library

About the CrossWorks Fixed Point Library

The *CrossWorks Fixed Point Library* is a work-in-progress that is intended to provide a complete fixed-point library for CrossWorks.

Architecture

The *CrossWorks Fixed Point Library* is one part of the *CrossWorks Target Library*. Many of the low-level functions provided by the target library are built using features of the *CrossWorks Tasking Library* for multi-threaded operation.

Delivery format

The *CrossWorks Fixed Point Library* is delivered in object form.

Feedback

This facility is a work in progress and may undergo rapid change. If you have comments, observations, suggestions, or problems, please feel free to air them on the **CrossWorks Target and Platform API** discussion forum.

License

The following terms apply to the Rowley Associates Fixed Point Library.

General terms

The source files and object code files in this package are not public domain and are not open source. They represent a substantial investment undertaken by Rowley Associates to assist CrossWorks customers in developing solutions using well-written, tested code.

Object Code Commercial License

If you hold a paid-for Object Code Commercial License for this product, you are free to incorporate the object code in your own products without royalties and without additional license fees. This Library is licensed to you PER DEVELOPER and is associated with a CrossWorks Product Key which, when combined, forms the entitlement to use this library. You must not provide the library to other developers to link against: each developer that links with this Library requires their own individual license.

<ahrs.h>

Overview

Attitude and heading reference system.

Two implementations of the AHRS algorithms are provided, one in floating point form and one in fixed point form. The API for both is identical, you simply choose whether to use floating or fixed at initialization time.

Typically, the fixed AHRS is fast on both Cortex-M3 and Cortex-M4 processors. For processors that have hardware floating point, there is little to no gain in choosing the fixed point implementation.

API Summary

Types	
AHRS_STATE_t	AHRS state
Setup	
ahrs_fixed_init	Initialize floating-point AHRS
ahrs_float_init	Initialize floating-point AHRS
Parameters	
ahrs_get_feedback	Get the long-term feedback factor
ahrs_set_feedback	Set the long-term feedback factor
Operation	
ahrs_get_orientation	Recover AHRS orientation estimation
ahrs_update	Update AHRS with new sensor measurement

AHRS_STATE_t

Synopsis

```
typedef struct {
  const  __AHRS_METHODS_s *__methods;
} AHRS_STATE_t;
```

Description

AHRS_STATE_t contains the internal state of the AHRS and all fields are considered private to the driver and inaccessible to the user. The API-level functions alter the state and inquire data, please do not directly reference the state!

ahrs_fixed_init

Synopsis

```
void ahrs_fixed_init(AHRS_STATE_t *self);
```

Description

ahrs_fixed_init initializes the AHRS reference system **self**. Once initialized, all computations that update the AHRS state are performed in fixed point.

Thread Safety

ahrs_fixed_init is thread safe.

ahrs_float_init

Synopsis

```
void ahrs_float_init(AHRS_STATE_t *self);
```

Description

ahrs_float_init initializes the AHRS reference system **self**. Once initialized, all computations that update the AHRS state are performed in floating point.

Thread Safety

ahrs_float_init is thread safe.

ahrs_get_feedback

Synopsis

```
float ahrs_get_feedback(AHRS_STATE_t *self);
```

Description

ahrs_get_feedback reads the long-term feedback factor.

Thread Safety

ahrs_get_feedback is thread safe.

ahrs_get_orientation

Synopsis

```
void ahrs_get_orientation(AHRS_STATE_t *self,
                          float *q);
```

Description

ahrs_get_orientation assigns the current orientation of the AHRS **self** to **q**. **q** is a pointer to an array of four floating values that are assigned the orientation in quaternion form, (*w*, *x*, *y*, *z*).

Thread Safety

ahrs_get_orientation is thread safe.

ahrs_set_feedback

Synopsis

```
void ahrs_set_feedback(AHRS_STATE_t *self,
                       float feedback);
```

Description

ahrs_set_feedback sets the long-term feedback factor, to correct the estimated orientation from the magnetometer and accelerometers.

The feedback factor **feedback** should be a value in the range from 0.0 to 0.5. With a factor of zero, any magnetometer and accelerometer data presented using **ahrs_update** is ignored and only the gyroscope data used to update the orientation estimation. In this case, there is no correction for long-term drift.

The feedback factor is set to 0.1 by the initialization functions.

Larger values of the feedback factor will make the initial orientation estimation converge faster to a fixed state. We have found that it is best to set the feedback factor to 0.3 for the first 200 samples or so, to quickly estimate orientation from the accelerometer and magnetometer, and then set the feedback factor to 0.1 for subsequent samples.

Thread Safety

ahrs_set_feedback is thread safe.

ahrs_update

Synopsis

```
void ahrs_update(AHRS_STATE_t *self,
                 float period,
                 float *data);
```

Description

ahrs_update updates the orientation estimation in the AHRS **self**. The elapsed time between the previous update and this update is passed as **period** seconds. The data for this update is passed in **data**:

- `data[0]`...`data[2]` hold the accelerometer X, Y, and Z measurements. Typically these are in *g* and come direct from a `CTL_ACCELEROMETER_t` object. However, the units for the accelerometer do not matter as the data are normalized before processing.
- `data[3]`...`data[5]` hold the spin rates about the X, Y, and Z axes in radians per second. Typically these come from a `CTL_GYROSCOPE_t` object, converted from degrees/second to radians/second, and passed into **ahrs_update**.
- `data[6]`...`data[8]` hold the field strength readings for the X, Y, and Z axes. Typically these come from a `CTL_MAGNETOMETER_t` object.

If the magnetometer measurement is not available and you wish to fuse only accelerometer and gyroscope, you must set the magnetometer values to an IEEE not-a-number (NaN) which is defined as `NAN` in `<math.h>`. If the accelerometer measurement is invalid (i.e. any axis measurement is NaN or all axis measurements are zero), the AHRS orientation estimation is not updated.

Note

The data held in **data** is modified during AHRS update.

Thread Safety

ahrs_update is thread safe.