Inertial Measurement Unit (IMU) Core
Reference Manual

by
Simeon Symeonidis

11/13/2018
revision 1.0

Overview

This reference manual documents the compiler options and application programming interface (API) to facilitate IMU core library integration into an embedded system or host application. For an easy to understand example, reference displayIMU/csv/csv_process.c. For an example on IMU calibration and instrumentation, review the displayIMU/display directory.

Compiler Options

displayIMU/setenv.sh defines environment variables needed for configuring the displayIMU code base. One significant design consideration is to enable a multi-threaded environment. The allows for a low-latency, non-blocking sensor interface and a asynchronous application interface. Currently, only the pthreads library is supported. To enable, set IMU_USE_PTHREAD to be non-zero. If the software needs to be modified to support a different multi-threaded library, the mutex code in IMU_thrd needs to be update along with the thread creation/deletion code in IMU_engn, IMU_pnts, and IMU_calb. If multi-threaded support is enable, a sensor first-in, first-out (FIFO) buffer is required, especially given sensors operating a different frequencies. Queue size can be set using IMU_ENGN_QUEUE_SIZE. It is recommended to set this between 3 and 5. If there are questions regarding how on how to adjust it, one can check the return value of the IMU_engn_datum function to determine how deep the queue is at a specified time. If multi-threaded support is not enabled, this should be set to zero.

The displayIMU project supports multiple instances, each containing their own configuration and state. The maximum number of instances can be set using IMU_MAX_INST. All memory within the core is statically allocated: by keeping the max instance count to a minimum, one can save memory resources. IMU_TYPE is used to specify the sensor data type. The nominal type is a signed short int or int16_t, but their may be scenarios where high-precision, i.e. 32-bit data is available, or the data is normalized a priory, i.e. floating-point. Internally, sensor data is type-casted to 32-bit floating-point values, but there is opportunity to optimize certain software parts to fixed precision. IMU_CALB_SIZE specifies the maximum number of point for a user-initiated or factory calibration. For nominal conditions, the max number of points for a calibration should not exceed 12. There are IMU_pnts hooks to store points and calls to reach back into the buffer. Its depth can be adjusted via IMU_PNTS_SIZE. This logic has not been tested and its recommended to keep this set to 1.

Core API

To initialize the IMU core, invoke IMU_engn_init(IMU_engn_type, uint16_t *id). The first parameter indicates which modules/subsystems are running. Use IMU_engn_calb_full to specify that IMU_core, IMU_rect, IMU_pnts, IMU_stat, and IMU_calb is enabled. A different IMU_engn_type should only be considered if there is limited processing resources or functions are being reproduced elsewhere. The function can be called several times, creating new IMU instances. Each instance will generate a unique instance handle (id). To configure the engine and its modules, use IMU_engn_load(uint16_t id, const char* filename, and IMU_engn_system). IMU_engn_system should be IMU_engn_self and if the json file configFileCore, configFileRect, configFilePnts, configFileStat, and configFileCalb parameters are set, will automatically read all configuration files. Last, IMU_engn_start() needs to be called to reset modules and enable the sensor FIFO.

Sensor data is injected via IMU_engn_datum(uint16_t id, IMU_datum*) or IMU_engn_data3(uint16_t id, IMU_data3*). The IMU_datum is for asynchronous data and IMU_data3 for synchronous data (see IMU_type.h for type definition). The time field, t, has a least significant bit (LSB) of 10us and the data

field(s) are the type specified by IMU_TYPE. Orientation and translational acceleration estimates can be accessed through IMU_engn_getEstm(uint16_t id, uint32_t t, IMU_engn_estm*). The isTran, isRef, isAng fields determine which elements of the IMU_engn_estm structure gets populated. If a reference is enabled, i.e. isRef is non-zero, IMU_engn_setRef(uint16_t id, float *ref) can manually set the system orientation reference. Otherwise, IMU_engn_setRefCur(uint16_t id) can set the reference orientation to the current estimate.

Calibration functions are available to populate IMU_rect correction configuration fields and IMU_core figure-of-merit (FOM) configuration fields. To initiate a manual, fixed-point calibration, invoke the IMU_engn_calbStart(uint16_t id, IMU_calb_mode, void*). By default, the calibration coefficients will automatically be saved, but if tthis is overwritten, call IMU_engn_calbSave(uint16_t id). To initiate a calibration touch-up via continuously updated stats use IMU_enng_calbStat(uint16_t id). If accuracy does not improve after calibration, IMU_engn_calbRevert(uint16_t id) can restore it to the previous configuration state. To provide a more complex user interface and to generate hooks for factory calibration procedures, function and variable pointers are available to automatically call functions upon specified events. To specify a function handler when the line-of-sight (LOS) has been stable enough to create a point use IMU_engn_setStableFnc(uint16_t id, void (*fnc)(IMU_PNTS_FNC_ARG), void*). To specify a function handle when a stable point has transition to moving use IMU_engn_setBreakFnc (uint16_t id, void (*fnc)(IMU_PNTS_FNC_ARG), void*). To specify a function handler when all the points for a manual calibration has been collected call IMU_engn_setCalbFnc(uint16_t id, void (*fnc) (IMU_CALB_FNC_ARG), void*). If configurated for a multi-thread environment, these function will be called within their own threads, minimizing latency impact.

Finally, instrumentation is available for debugging and tooling. If the IMU_engn isSensorStruct config field is set, a copy of the lastest gyroscope, accelerometer, and magnetometer raw data, corrected data, filtered data, and figure-of-merit can be read. To get this handle, use IMU_engn_getSensor(uint16_t id, IMU_engn_sensor**). Having access to a module's state can also be helpful. For example, IMU_stat state contains info on sensor magnitudes, biases, and dot products. IMU_engn_getState(uint16_t id, IMU_engn_system, IMU_union_state*) can be used to get a pointer to this structure.