Sensor Board

All about the sensor board

- Overview
- STM32CubeMX Sensor Configuration
- Sensor Basics Utility Library
- Architecture
- Configuration
- pH Sensor
- IMU
- Load Cell
- Pressure Sensor
- Testing
- Reference

Overview

The Sensor Board is an embedded system that acquires environmental and process data from multiple sensors and actuates the sampling hardware (water pump). It integrates water quality (pH), motion (IMU), force (load cells), pressure (FSR pads), and water flow measurement, and transmits all data over Ethernet using Protocol Buffer encoding. It also receives control packets (for example pump commands) over the same link.

Implementation Status Disclaimer: The sensor board code was only partially tested and never fully implemented on hardware. Due to time constraints and other technical issues faced by the 2025-2026 team, several drivers remain compile gated or placeholder (pH ADC path, pressure ADC path, IMU bus access, flow sensor EXTI line). The load cell (HX711), pump (PWM), networking and protobuf pipeline are implemented in firmware; end to end validation on the assembled board was not completed.

Hardware Platform

Item	Value
Microcontroller	STM32H753ZI (NUCLEO-H753ZI board), ARM Cortex-M7 (480 MHz capable, currently clocked at 64 MHz, see STM32CubeMX Sensor Configuration)
Real-Time OS	FreeRTOS with CMSIS-RTOS V2
Ethernet	LAN8742 PHY, RMII, LwIP stack (static IP, no DHCP)
Serial logging	ST-Link VCP (USART3, 115200 baud)
FreeRTOS heap	64 KB (configTOTAL_HEAP_SIZE = 65536)
Message encoding	nanopb (Protocol Buffers), definitions in the ERC-Protobufs submodule

Key Board Features

- Single main task polls all sensors in one loop (5 second interval)
- Network integration via UDP/Ethernet with protobuf envelopes (PBEnvelope)
- Inbound packet dispatcher for control signals (pump command handler actuates hardware)
- Real-time logging to UART (115200 baud) with uniform per-sensor status lines
- MAC address filtering for selective communication
- Static ARP entry for the destination board, re-added on link up

- LED status indicators (Green, Blue, Red toggled each loop)
- Heap monitoring with critical threshold alert (below 4096 bytes free pauses the loop)

Integrated Sensors and Actuators

Device	Model / Interface	Count	Firmware status
pH Sensor	DFRobot SEN0161, analog ADC	1	Driver complete, ADC path compile gated (PH_SENSOR_USE_ADC), poll returns RESULT_ERR_UNIMPLEMENTED until an ADC is enabled in CubeMX
IMU	TBD, I2C1 (PB8/PB9)	1	Data structure, math and validation helpers implemented; hardware poll is a placeholder
Load Cell (Weight)	HX711 24-bit ADC, GPIO bit-bang	2	Implemented, auto-tare on init, scale calibration API
Pressure (FSR)	Analog force sensing resistor pads, ADC	2	Driver complete, ADC path compile gated (PRESSURE_USE_ADC), poll returns RESULT_ERR_UNIMPLEMENTED until an ADC is enabled
Flow Sensor	FM-PS2216 (5.5 pulses/ml), GPIO EXTI pulse counting on PA4	1	Implemented; EXTI4 must still be enabled in CubeMX for pulses to be counted
Water Pump	Grothen 12 V DC mini peristaltic pump, single MOSFET on TIM3 CH3 PWM (PC8)	1	Implemented, open loop, unidirectional; health cross-checked against the flow sensor

Check this out for compiling code and more about project structure- Project Structure

Related Pages

- STM32CubeMX Sensor Configuration
- Sensor Board Utility Library
- Architecture
- Configuration
- pH Sensor

- Load Cell (Weight Sensor)
- Pressure Sensor
- Testing
- Reference

STM32CubeMX Sensor Configuration

This page documents the current STM32CubeMX configuration for the Sensor Board firmware and explains how to extend it for the remaining sensor interfaces (ADC, EXTI, I2C device setup) in a way that is safe for code generation.

- **IOC File:** components/sensor_board/firmware/firmware.ioc
- **Generated HAL Init Files:** components/sensor_board/firmware/Core/Src/
- **Application Entry:** src/sensor_board/main.c (MainTask, entry set to "As external" in FreeRTOS tab)

Current CubeMX Snapshot

Item	Value
MCU	STM32H753ZITx (NUCLEO-H753ZI)
CubeMX Version	6.15.0
STM32Cube FW Package	STM32Cube FW_H7 V1.12.1
Build system	PlatformIO (env:sensor_board), stm32cube framework

Enabled CubeMX Components

- CORTEX_M7 (I-Cache and D-Cache enabled, MPU configured)
- DMA (DMA1 Stream 0)
- ETH (RMII mode)
- FREERTOS (CMSIS-RTOS V2, defaultTask plus mainTask)
- I2C1 (fast mode, for the IMU on PB8/PB9)
- LWIP (static IP, DHCP disabled, static ARP entries enabled)
- TIM3 (PWM Generation CH3, pump MOSFET gate)
- TIM7 (base timer)
- USART1 (asynchronous)
- SYS/NVIC/RCC base platform configuration (HAL timebase on TIM6)

Clock and Core Setup

Clock Configuration (from IOC)

Parameter	Value
HSE crystal value	8 MHz
SYSCLK	64 MHz
APB1	64 MHz

Note: the STM32H753 is capable of 480 MHz, but the current IOC runs the core at 64 MHz. Raising the clock requires PLL configuration in the Clock Configuration tab and re-validation of the Ethernet and FreeRTOS timing.

Cortex-M7 / MPU

- Instruction cache: enabled
- Data cache: enabled
- MPU region at 0x30000000, size 32 KB, full access, TEX level 1 (non-cacheable region for the Ethernet DMA descriptors and LwIP heap)
- LwIP RAM heap pointer: 0x30004900, MEM_SIZE 16 KB

Pinout and Peripheral Mapping

Ethernet (RMII)

Signal	Pin
ETH_REF_CLK	PA1
ETH_MDIO	PA2
ETH_CRS_DV	PA7
ETH_MDC	PC1
ETH_RXD0	PC4
ETH_RXD1	PC5
ETH_TX_EN	PG11
ETH_TXD0	PB12
ETH_TXD1	PB13

Sensor and Actuator Pins (labeled in IOC)

Label	Pin	Mode	Used by
FLOW_SENSOR	PA4	GPIO Input (EXTI4 not yet enabled)	Flow sensor pulse counting
WEIGHT_INPUT_1	PA5	GPIO Input	HX711 unit 0 DOUT
WEIGHT_INPUT_2	PA6	GPIO Input	HX711 unit 1 DOUT
WEIGHT_CLOCK_1	PC7	GPIO Output	HX711 unit 0 SCK
WEIGHT_CLOCK_2	PB5	GPIO Output	HX711 unit 1 SCK
IMU_I2C_Clock	PB8	I2C1_SCL (pull-up)	IMU
IMU_I2C_Data	PB9	I2C1_SDA (pull-up)	IMU
WATER_PUMP_PWM	PC8	TIM3_CH3 (PWM)	Pump MOSFET gate
PH_ANALOG_DATA	PD14	GPIO_Analog	pH sensor (see warning below)
FORCE_ANALOG_DATA_1	PD15	GPIO_Analog	Pressure/FSR sensor 0 (see warning below)
FORCE_ANALOG_DATA_2	PF3	GPIO_Analog	Pressure/FSR sensor 1 (PF3 = ADC3_INP5)
STEPPER_MOTOR_1..4	PD7, PD6, PD5, PD4	GPIO Output	Reserved for the sampling stepper

Warning: on the STM32H753, PD14 and PD15 have NO ADC function. The pH analog input and FORCE_ANALOG_DATA_1 must be moved to ADC-capable pins (for example PA0, PC0, or PF3-class pins) before the analog drivers can be enabled. PF3 maps to ADC3_INP5 and can be used as is.

Serial / COM

Signal	Pin	Note
USART1_TX / USART1_RX	PA9 / PA10	Spare asynchronous UART
USART3_TX / USART3_RX	PD8 / PD9	NUCLEO ST-Link VCP path, used by the logging system (hcom_uart[COM1])

RTOS Tasks and Interrupts

FreeRTOS Tasks (CMSIS V2)

Task	Priority	Stack (words)	Entry

defaultTask	24	128	StartDefaultTask (generated)
mainTask	39	2048	MainTask, "As external" (defined in src/sensor_board/main.c)

FreeRTOS heap: configTOTAL_HEAP_SIZE = 65536 (64 KB).

Interrupt Priorities (key entries)

IRQ	Priority	Notes
ETH_IRQn	5	Ethernet / LwIP path
DMA1_Stream0_IRQn	5	DMA
USART1_IRQn	5	Spare UART
EXTI15_10_IRQn	5	External interrupt group (user button)
TIM6_DAC_IRQn	15	HAL tick timebase (TIM6)
SysTick / PendSV	15	FreeRTOS kernel

Sensor Interface Status

What Is Already Configured in CubeMX

- Networking stack (LwIP) and RMII pinout
- FreeRTOS scaffolding with the external MainTask
- I2C1 bus for the IMU with pull-ups on PB8/PB9
- TIM3 CH3 PWM output for the pump
- GPIO pins and labels for the HX711 load cells and the flow sensor

What Is Not Yet Modeled in CubeMX (TO DO once sensors are retrieved and assembled)

- ADC peripherals and channels for the analog sensors (pH, pressure/FSR); PD14 and PD15 must also be moved to ADC-capable pins
- EXTI4 rising-edge interrupt for the flow sensor pin PA4 (until enabled, flow always reads 0)
- IMU device bring-up on I2C1 (the poll function is a placeholder)

Important: the analog sensor drivers are compile gated (PH_SENSOR_USE_ADC, PRESSURE_USE_ADC) so the firmware links without hardware. When bringing up physical sensors, add the corresponding CubeMX peripherals first, then enable the build flags and bind the generated handles.

Recommended Workflow

1. Open components/sensor_board/firmware/firmware.ioc in STM32CubeMX.
2. Add the required peripheral for the target sensor (ADC channel, EXTI line, or I2C settings).
3. Assign and lock pins in the Pinout view; avoid overlap with RMII and COM pins.
4. Configure clocks for new peripherals in Clock Configuration.
5. Set NVIC priorities for new ISR sources so Ethernet and RTOS timing remain stable.
6. Generate code with Keep User Code enabled.
7. Rebuild using PlatformIO and validate startup and sensor polling.

Conflicts To Check Before Saving the .ioc File

- No conflict with ETH RMII pins (PA1, PA2, PA7, PC1, PC4, PC5, PG11, PB12, PB13)
- No conflict with the debug/COM path (PA9/PA10 and PD8/PD9)
- No conflict with oscillator pins (PH0, PH1, PC14, PC15)
- No conflict with the existing sensor labels (PA4, PA5, PA6, PB5, PB8, PB9, PC7, PC8, PD4..PD7, PD14, PD15, PF3)

Related Pages

- Configuration
- Reference
- Sensor Board Utility Library

Sensor Basics Utility Library

The Sensor Basics utility library provides small validation and conversion helpers shared by the sensor drivers and the main application. These are basic features that could be used if required, but were made in "spare time".

Source Code Location

Files:

- `components/sensor_board/sensor_basics/sensor_basics.h` (function declarations and documentation)
- `components/sensor_board/sensor_basics/sensor_basics.c` (implementation)

Dependencies:

- `result.h` (standard result/error code definitions and the TRY macro)
- `stdint.h` (integer type definitions)

pH Validation

validate_ph_value()

Validates that a pH value is within the acceptable range (0 to 14).

```
result_t validate_ph_value(float ph_value) {
    if (ph_value >= 0.0f && ph_value <= 14.0f) {
        return RESULT_OK;
    }
    return RESULT_ERR_INVALID_DATA;
}
```

Return values: RESULT_OK when the value is within range, RESULT_ERR_INVALID_DATA otherwise.

Note: ph_sensor_update() already clamps its output to 0..14, so this check only fails for values that bypass the driver (for example raw values received over the network).

IMU (Accelerometer) Validation

validate_accelerometer_value()

Validates one accelerometer axis. Valid range is -160.0 to +160.0 m/s², which corresponds to a typical ±16 g sensor range.

```
result_t validate_accelerometer_value(float accel_value) {
    if (accel_value >= -160.0f && accel_value <= 160.0f) {
        return RESULT_OK;
    }
    return RESULT_ERR_INVALID_DATA;
}
```

validate_imu_data()

Validates all three accelerometer axes at once using the TRY macro for early return on the first invalid axis.

```
result_t validate_imu_data(float accel_x, float accel_y, float accel_z) {
    TRY(validate_accelerometer_value(accel_x));
    TRY(validate_accelerometer_value(accel_y));
    TRY(validate_accelerometer_value(accel_z));
    return RESULT_OK;
}
```

Note: the IMU driver has its own richer validators (imu_validate_accelerometer_range, imu_validate_gyroscope_range, imu_validate_magnetometer_range) which the main loop uses. See the imu component.

Conversion Functions (declared, currently inactive)

The header declares four unit conversion helpers. Their implementations exist in sensor_basics.c but are commented out, so linking against them fails until they are re-enabled.

```
/* Temperature: F = C * 9/5 + 32,  C = (F - 32) * 5/9 */
result_t celsius_to_fahrenheit(float celsius, float *fahrenheit);
result_t fahrenheit_to_celsius(float fahrenheit, float *celsius);

/* Pressure: psi = bar * 14.5038,  bar = psi / 14.5038 */
result_t bar_to_psi(float bar, float *psi);
result_t psi_to_bar(float psi, float *bar);
```

Each returns RESULT_OK on success or RESULT_ERR_INVALID_ARG when the output pointer is NULL.

Implementation Status

Currently implemented (active):

- validate_ph_value()
- validate_accelerometer_value()
- validate_imu_data()

Currently commented out (inactive):

- celsius_to_fahrenheit()
- fahrenheit_to_celsius()
- bar_to_psi()
- psi_to_bar()

Note: earlier drafts of this page documented GPS validation helpers (latitude, longitude, HDOP, satellite count). These functions do not exist in the current library since there is no GPS driver on the sensor board.

Error Handling Pattern

All validation functions follow the same pattern used across the firmware:

```
if (validate_ph_value(ph_value) == RESULT_OK) {
    diagnostics.ph_sensor.state = SensorState_SENSOR_OPERATING;
    diagnostics.ph_sensor.error_code = PHErrorCode_PH_NO_ERROR;
} else {
    diagnostics.ph_sensor.state = SensorState_SENSOR_ERROR;
    diagnostics.ph_sensor.error_code = PHErrorCode_PH_INVALID_DATA;
```

```
}
```

Testing

Test suite location: `test/sensor_board/test_sensor_basics/`

```
# Run only the sensor_basics tests
pio test -e sensor_board -f test_sensor_basics

# Run with verbose output
pio test -e sensor_board -f test_sensor_basics -v
```

Current coverage: accelerometer boundary values (±160.0 accepted, ±160.1 rejected) and multi-axis combination. The temperature and pressure conversion tests exist in the file but are commented out together with their implementations.

Architecture

Complete system overview: FreeRTOS task model, the sensor polling loop, protobuf encoding, UDP transmission, inbound packet dispatch, and the memory layout. All application logic lives in a single FreeRTOS task (MainTask) defined in src/sensor_board/main.c.

Initialization Sequence

Phase 1: Hardware Setup (init_board, before the kernel starts)

```
void init_board() {
  MPU_Config_wrapper();
  SCB_EnableICache();
  SCB_EnableDCache();
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  /* NOTE: no threads here, kernel not initialized yet.
   * osKernelInitialize() is called by cubemx_main.c afterwards. */
}
```

Phase 2: Driver Initialization (start of MainTask)

1. BSP LEDs (Green, Blue, Red)
2. Logging over the ST-Link VCP UART (LOG_init(&hcom_uart[COM1]), 115200 baud)
3. IMU (imu_sensor_init)
4. pH sensor (ph_sensor_init with 3.3 V reference)
5. Two HX711 load cells with their GPIO map (PA5/PC7 and PA6/PB5), each powered up and auto-tared
6. Two pressure/FSR sensors (pressure_sensor_init)
7. Flow sensor (flow_sensor_init, pulse counting via EXTI callback)
8. Pump (pump_init on TIM3 CH3 PWM, then commanded to 50 percent and enabled as a startup default)

Phase 3: Communication Setup

1. ETH_init with static IP 192.168.0.111, netmask 255.255.255.0, gateway 192.168.0.1 and a link status callback that re-adds the static ARP entry when the link comes up
2. MAC address filtering for three allowed source MACs (ETH_setup_MAC_address_filtering)
3. Two statically allocated prioritised UDP transmit queues (80 entries each)
4. Packet dispatcher registration for five inbound message types (pH, IMU, load cell, pressure, pump)
5. ETH_udp_init(2, send_queues, DispatchPacket) and a static ARP entry for the destination board 192.168.0.222

Phase 4: Main Loop

The loop runs forever with a 5000 ms period (MAIN_TASK_DELAY_MS). Each iteration:

1. Read free heap; if below 4096 bytes log CRITICAL and sleep 10 s instead of polling
2. Toggle the three LEDs (visual heartbeat)
3. Build a SensorBoardDiagnostics struct (state OPERATING)
4. Poll pH, IMU, then load cells and pressure sensors (index loop over both units)
5. Poll the flow sensor (rate computed from pulses counted by the EXTI ISR since the last poll)
6. Build the pump status from commanded state, cross-checked against measured flow
7. Wrap each sensor message in a PBEnvelope and send it as a UDP datagram to the sample board
8. osDelay(MAIN_TASK_DELAY_MS)

Protobuf Encoding and UDP Transmission

Every outbound message is one PBEnvelope with a oneof payload. Encoding uses nanopb via pb_message_encode, which allocates a heap buffer that is freed after sending.

```
static void udp_send_envelope(uint8_t dest_ip[4], PBEnvelope *env) {
  if (!sendUDP) { return; }   /* transmit gate, false by default */
  uint8_t *encoded = NULL;
  size_t size = 0;
  result_t result = pb_message_encode(env, PBEnvelope_fields, &encoded, &size);
  if (result == RESULT_OK) {
    ETH_udp_send(dest_ip, PORT, encoded, (uint16_t)size, 1);
  }
  free(encoded);
}
```

Important: the static flag `sendUDP` in main.c is currently `false`, so envelope encoding and transmission are skipped entirely. Set it to true to actually transmit. There is a similar development flag `skip_sensor_polling` (currently false) that disables all sensor polling when true.

Inbound Packet Dispatcher

Received UDP packets are decoded by the shared packet dispatcher (components/common/packet_dispatcher). Handlers are registered with PACKET_HANDLER_CONFIG_STATIC per envelope payload tag:

Envelope tag	Handler	Behavior
ph_info	handle_sensor_ph_info	Log only
imu_info	handle_sensor_imu_info	Log only
load_cell_info	handle_sensor_load_cell_info	Log only
pressure_info	handle_sensor_pressure_info	Log only
pump_info	handle_sensor_pump_command	Actuates the pump: applies enabled, direction, and speed_percent to the hardware

Sensor Status Model

SensorState (operating state)

Code	Meaning
SENSOR_IDLE	Not connected, not implemented, or intentionally off
SENSOR_OPERATING	Normal operation, valid data
SENSOR_ERROR	Communication failure or invalid data

SensorStatus (connection status)

Code	Meaning
STATUS_OK	Healthy
STATUS_DISCONNECTED	No hardware detected (poll returned UNIMPLEMENTED or COMMS)
STATUS_ERROR	Unexpected failure

STATUS_INITIALIZING	Warming up (flow sensor first sample window)

Poll Result Mapping

handle_sensor_poll_result() maps driver results uniformly:

- RESULT_ERR_UNIMPLEMENTED or RESULT_ERR_COMMS: state IDLE, status DISCONNECTED (sensor not connected or driver not wired to hardware yet)
- Any other non-OK result: state ERROR, status ERROR
- RESULT_OK: caller then validates the data and picks OPERATING or ERROR plus a driver-specific error code

Every sensor produces one uniform log line per loop: `name | STATUS | STATE | detail`.

Pump Health Cross-Check

The pump is an open loop actuator (no current sense or fault line), so firmware cannot directly detect a connected pump. The only on-board proof that fluid is moving is the inline flow sensor, so the main loop derives pump status from it:

- Not initialised: ERROR / ERROR
- Commanded off (disabled or 0 percent): IDLE / OK (healthy, intentionally off)
- Commanded on and flow detected: OPERATING / OK
- Commanded on and no flow: OPERATING / DISCONNECTED (pump absent, dry, stalled, or the flow sensor is not installed)

Memory Layout

Region	Use
FreeRTOS heap, 64 KB	Task stacks, queues, protobuf encode buffers (malloc/free per message)
0x30000000, 32 KB, MPU non-cacheable	Ethernet DMA descriptors and buffers
0x30004900, 16 KB	LwIP RAM heap (MEM_SIZE)
Static queues	Two UDP send queues, 80 entries each, allocated at compile time (xQueueCreateStatic)

Error Handling Strategy

- Drivers never crash the loop: missing hardware degrades to IDLE/DISCONNECTED and the loop continues
- Each sensor unit reports independently (separate envelope, separate error code enum)
- Heap exhaustion protection: below 4096 bytes free, polling pauses for 10 s per iteration
- Encode failures are logged with result_to_short_str / result_to_desc_str and the buffer is freed in all paths

Configuration

Compile-time parameters, runtime flags, sensor calibration setup, network addressing, and performance tuning options for the sensor board firmware.

Main Loop Timing

```
/* src/sensor_board/main.c */
#define MAIN_TASK_DELAY_MS 5000   /* poll + transmit interval */
```

All sensors are polled and transmitted once per interval. Lowering this value increases network traffic and heap churn (one encode allocation per message).

Runtime Flags (main.c)

Flag	Default	Effect
sendUDP	false	When false, udp_send_envelope() returns immediately: nothing is encoded or transmitted. Set to true to enable network output.
skip_sensor_polling	false	When true, the pH/IMU/load cell/pressure polling block is skipped (flow and pump still run).

Heap Management

```
uint32_t free_heap = xPortGetFreeHeapSize();
if (free_heap < 4096U) {   /* critical threshold, was 8192U */
    LOGE(TAG, "CRITICAL: Low heap! Free: %lu bytes", free_heap);
    osDelay(10000);
    continue;
}
```

- FreeRTOS heap size: 65536 bytes (configTOTAL_HEAP_SIZE, set in CubeMX)
- Critical threshold: 4096 bytes free

- LwIP heap: 16 KB at 0x30004900 (separate from the FreeRTOS heap)

Network Configuration

All addresses live in `components/common/networking_constants/ip_mac_constants.h`:

```
#define NETWORK_IP       {192, 168, 0, 111}                    /* this board */
#define NETWORK_MAC      {0x00, 0x80, 0xe1, 0x00, 0x00, 0x00}

#define SAMPLE_BOARD_IP  {192, 168, 0, 222}                    /* UDP destination */
#define SAMPEL_BOARD_MAC {0x00, 0x43, 0x23, 0xee, 0x21, 0x64}

#define GATEWAY          {192, 168, 0, 1}
#define NETMASK          {255, 255, 255, 0}

#define PORT 1500                                              /* UDP port */
```

Changing the Destination Address or Port

1. Edit SAMPLE_BOARD_IP / SAMPEL_BOARD_MAC (used both for sending and for the static ARP entry) or PORT in ip_mac_constants.h.
2. Rebuild; no other file references the raw addresses.

MAC Address Filtering

Three source MACs are allowed through the hardware filter, configured in MainTask:

```
int mac1[6] = {0x11, 0x22, 0x33, 0x44, 0x55, 0x66};
int mac2[6] = {0x12, 0x23, 0x34, 0x45, 0x56, 0x67};
int mac3[6] = {0x90, 0x2e, 0x16, 0xbe, 0x1b, 0x33};
ETH_setup_MAC_address_filtering(mac1, mac2, mac3);
```

UDP Transmit Queues

```
#define SENSOR_SEND_QUEUE_SIZE 80   /* entries per priority queue, x2 queues */
```

Sensor Hardware Build Flags

The analog drivers are compile gated so the firmware links without ADC hardware. Add these to build_flags in platformio.ini once the corresponding CubeMX peripherals exist:

Flag	Effect
-D PH_SENSOR_USE_ADC	Enables the pH ADC read path (optional: -D PH_SENSOR_ADC_HANDLE=hadc1, -D PH_SENSOR_ADC_MAX=65535, -D PH_SENSOR_ADC_TIMEOUT_MS=100)
-D PRESSURE_USE_ADC	Enables the pressure/FSR ADC read path; bind each unit with pressure_sensor_init_hw()
-D PUMP_MAX_RPM_EST=100	Override the pump RPM estimate used when converting duty cycle to speed_rpm
-D CONFIG_LOG_LEVEL=LOG_INFO	Log verbosity (already set in platformio.ini)

Sensor Calibration Configuration

pH Sensor

```
/* components/sensor_board/ph/ph_sensor.h */
#define PH_SAMPLE_COUNT   40     /* averaging buffer size */
#define PH_DEFAULT_SLOPE  3.5f   /* SEN0161 default: pH = 3.5 * V + offset */

/* runtime calibration */
ph_sensor_calibrate(&ph_sensor, offset, slope);
ph_sensor_reset_calibration(&ph_sensor);   /* offset 0, slope 3.5 */
```

Load Cell (HX711)

```
/* components/sensor_board/load_cell/load_cell_sensor.h */
#define LOAD_CELL_DEFAULT_N_PER_COUNT 1.0f  /* passthrough until calibrated */
#define LOAD_CELL_GAIN_PULSES         1U    /* 1 = ch A gain 128 */
#define LOAD_CELL_READY_TIMEOUT_MS    200U  /* wait for DOUT low */

/* runtime calibration */
load_cell_tare(&cell, 10);                 /* average 10 reads as zero */
load_cell_set_scale(&cell, n_per_count);   /* marks is_calibrated = true */
```

Pressure / FSR

```
/* kPa = voltage * scale_kpa_per_volt + offset_kpa */
pressure_sensor_set_calibration(&sensor, scale_kpa_per_volt, offset_kpa);
```

Flow Sensor

```
/* components/sensor_board/sampling/flow_sensor/flow_sensor.h */
#define FLOW_SENSOR_PULSES_PER_ML_X10 55U    /* FM-PS2216: 5.5 pulses/ml */
#define FLOW_SENSOR_SAMPLE_WINDOW_MS  1000U  /* rate calculation window */
#define FLOW_SENSOR_MAX_FLOW_ML_MIN   150U   /* plausibility clamp */
```

Changing the Sensor Poll Interval

1. Edit MAIN_TASK_DELAY_MS in src/sensor_board/main.c.
2. Keep it well above the slowest blocking read (HX711 ready wait can take up to 200 ms per cell).
3. The flow rate is computed per poll from the pulse count, so the interval also sets flow rate resolution.

Enabling / Disabling Sensors

- All sensor polling: skip_sensor_polling flag in main.c (flow and pump are outside this block)
- Individual analog sensors: leave their build flag unset; the driver returns RESULT_ERR_UNIMPLEMENTED and the sensor reports IDLE / DISCONNECTED without affecting the rest of the loop

pH Sensor

The pH sensor provides water quality measurement critical for environmental monitoring and anomaly detection.

Hardware Specifications

Parameter	Value
Model	DFRobot SEN0161 (Analog pH meter)
Interface	Analog ADC (compile gated, see Hardware Status below)
Reference Voltage	Configurable at init (3.3 V used in main.c; SEN0161 itself prefers a stable 5.0 V supply)
Board pin	PD14, labeled PH_ANALOG_DATA (must be moved, PD14 has no ADC function)
Measurement Range	0 to 14 pH units (clamped in software)
Accuracy	±0.1 pH @ 25°C (sensor datasheet)
Sample Averaging	40 samples, min and max excluded

Hardware Status

No ADC is enabled in CubeMX yet, so the ADC path is compile gated by `PH_SENSOR_USE_ADC`. Without that flag, `poll_ph_sensor()` returns RESULT_ERR_UNIMPLEMENTED and the sensor reports IDLE / DISCONNECTED (the firmware still links and runs). To enable:

1. In CubeMX enable an ADC (for example ADC1) and a channel on the pH input pin. PD14 (the current PH_ANALOG_DATA label) has NO ADC function on the STM32H753, so move the pH input to an ADC-capable pin (PA0, PC0, ...).
2. The SEN0161 is a 5 V board with output up to about 3 V. The STM32 ADC tops out at 3.3 V, so power or scale the board so its output never exceeds 3.3 V.
3. Build with `-D PH_SENSOR_USE_ADC` (optionally -D PH_SENSOR_ADC_HANDLE=hadc1, -D PH_SENSOR_ADC_MAX=65535).

Calibration Model

The sensor uses linear voltage-to-pH conversion (DFRobot SEN0161 formula):

```
pH = slope × Voltage + offset
Voltage = averaged_ADC / adc_max × reference_voltage
```

Default Parameters for SEN0161 @ 25°C:

- **Slope**: 3.5 (PH_DEFAULT_SLOPE)
- **Offset**: 0.0 by default, set via user calibration

The computed pH is clamped to the 0 to 14 range inside ph_sensor_update().

Data Structure

```
typedef struct {
    uint16_t raw_value;              /* averaged raw ADC value */
    float voltage;                   /* converted voltage */
    float ph_value;                  /* calculated pH (0-14), init 7.0 */
    float reference_voltage;         /* ADC reference */
    ph_calibration_t calibration;    /* { offset: float, slope: float } */
    uint16_t sample_buffer[40];      /* last 40 samples (PH_SAMPLE_COUNT) */
    uint8_t sample_index;            /* current position in buffer */
    uint8_t samples_collected;       /* samples collected so far (0-40) */
} ph_sensor_t;
```

Initialization & Usage

Initialize pH Sensor

```
ph_sensor_t ph_sensor;
ph_sensor_init(&ph_sensor, 3.3f);  /* 3.3 V reference, as in main.c */
```

Poll pH Sensor

```
result_t ph_result = poll_ph_sensor(&ph_sensor);

if (ph_result == RESULT_OK) {
    float ph_value, voltage;
```

```
    ph_sensor_get_value(&ph_sensor, &ph_value);
    ph_sensor_get_voltage(&ph_sensor, &voltage);
}
/* RESULT_ERR_UNIMPLEMENTED: ADC path not compiled in
 * RESULT_ERR_COMMS: HAL ADC start/conversion failed */
```

Manual Sample Addition

```
/* For manual sampling at regular intervals (e.g. every 20 ms) */
uint16_t adc_reading = 2048;
ph_sensor_add_sample(&ph_sensor, adc_reading);

/* Or feed a reading through the full pipeline (average + convert) */
ph_sensor_update(&ph_sensor, adc_reading, 4095);
```

Validation

```
result_t validate_ph_value(float ph_value);
/* Returns RESULT_OK if 0 <= ph_value <= 14
 * Returns RESULT_ERR_INVALID_DATA otherwise */
```

Sample Averaging Strategy

Parameter	Value
Sample Buffer Size	40 samples (PH_SAMPLE_COUNT)
Method	Circular buffer average, minimum and maximum values excluded (DFRobot sample code algorithm); simple average while fewer than 5 samples collected
Purpose	Noise filtering and stable readings

Averaging Algorithm

```
1. ADC sample added to circular buffer
2. Buffer averaged with min and max excluded
3. Averaged value converted to voltage
```

```
4. Voltage converted to pH via calibration (slope, offset)
5. pH clamped to 0-14
```

Two-Point Calibration Procedure

Step 1: Neutral Point (pH 7.0)

```
1. Short the BNC input or immerse the electrode in pH 7.0 buffer solution
2. Wait for a stable reading (~2 minutes)
3. Record the reported pH value
4. offset = 7.0 - recorded_value
```

Step 2: Slope Calibration (pH 4.0 or 10.0)

```
1. Immerse the electrode in a second known pH solution (pH 4.0 buffer)
2. Wait for a stable reading
3. Adjust the board gain potentiometer until the reading is 4.0,
   or compute: slope = (pH_reference - 7.0) / (V_reference - V_neutral)
4. Apply with ph_sensor_calibrate(&sensor, offset, slope)
```

Protobuf Message Format

```
message SensorBoardPHInfo {
    float ph_value;
    float voltage;
    SensorState state;
    PHErrorCode error_code;
}

enum PHErrorCode {
    PH_NO_ERROR = 0;
    PH_COMMUNICATION_FAILURE = 1;
    PH_INVALID_DATA = 2;
}
```

Error Handling (as in main.c)

```
if (ph_result == RESULT_ERR_UNIMPLEMENTED || ph_result == RESULT_ERR_COMMS) {
    /* Hardware not connected / ADC not enabled */
    diagnostics.ph_sensor.state = SensorState_SENSOR_IDLE;
    diagnostics.ph_sensor.error_code = PHErrorCode_PH_COMMUNICATION_FAILURE;
} else if (ph_result == RESULT_OK) {
    if (validate_ph_value(ph_value) == RESULT_OK) {
        diagnostics.ph_sensor.state = SensorState_SENSOR_OPERATING;
        diagnostics.ph_sensor.error_code = PHErrorCode_PH_NO_ERROR;
    } else {
        diagnostics.ph_sensor.state = SensorState_SENSOR_ERROR;
        diagnostics.ph_sensor.error_code = PHErrorCode_PH_INVALID_DATA;
    }
}
```

Integration Notes

- Single sensor instance in the main application, initialized with a 3.3 V reference
- Updates transmitted to the network at the main loop interval (5 seconds default), when the sendUDP flag is enabled
- Temperature compensation not implemented (assumes ~25°C)
- Because ph_sensor_update() clamps to 0-14, validate_ph_value() cannot fail on driver output; it protects against values from other sources
- Electrode response time: ~100-300 ms depending on pH change magnitude
- Unit tested on host: initialization defaults, voltage conversion, clamping, calibration (test/sensor_board/test_ph_sensor)

IMU

The Inertial Measurement Unit (IMU) provides three-axis acceleration, angular velocity, and magnetic field measurements for attitude determination and motion analysis.

Hardware

Parameter	Value
Device	Xsens Avior (Xbus protocol, MTi-1-series compatible pipe interface)
Interface	I2C1, pins PB8 (IMU_I2C_Clock) / PB9 (IMU_I2C_Data), internal pull-ups enabled
I2C address	0x6B (7-bit, MTi-1 series default, override with -D XSENS_I2C_ADDR_7BIT)
Output rate	100 Hz requested for accel/gyro/mag (XSENS_OUTPUT_RATE_HZ)
I2C timeout	100 ms (XSENS_I2C_TIMEOUT_MS)

Important: the pipe opcodes, default I2C address, and Xbus data identifiers used by the driver are the documented Xsens MTi-1-series values. The Avior is Xbus-compatible, but confirm these against the Avior datasheet (mtidocs.xsens.com) and override with -D build flags if your unit differs. The driver was never validated against physical hardware.

Communication Protocol (Xbus over I2C)

Xbus messages move through "pipe" opcodes used as an 8-bit register address:

- 0x03 ControlPipe: write Xbus command messages
- 0x04 PipeStatus: read 4 bytes, notification size (LE16) and measurement size (LE16)
- 0x06 MeasurementPipe: read a pending MTData2 measurement message

```
Xbus frame: [0xFA][0xFF][MID][LEN][DATA...][CHK]
CHK makes (BID + MID + LEN + DATA + CHK) & 0xFF == 0
```

On the first poll the driver configures the device once: GoToConfig, SetOutputConfiguration (acceleration + rate of turn + magnetic field at 100 Hz, float32), GoToMeasure. If configuration fails, poll returns RESULT_ERR_COMMS (device not responding on I2C).

Data Structure

```
typedef struct {
    float accel[3];            /* [X, Y, Z] acceleration, m/s² */
    float gyro[3];             /* [X, Y, Z] angular velocity, °/s
                                  (converted from rad/s by the driver) */
    float mag[3];              /* [X, Y, Z] magnetic field, Xsens arbitrary
                                  units (~1.0 = local Earth field), NOT µT */
    uint32_t timestamp;        /* Current reading timestamp (HAL_GetTick ms) */
    uint32_t last_timestamp;   /* Previous reading timestamp */
} imu_data_t;
```

Unit notes: the gyroscope values are converted from rad/s to °/s inside the driver. The Xsens magnetic field output is in arbitrary units where roughly 1.0 equals the local Earth field; it is stored as-is and must be scaled externally if µT are needed.

Initialization & Usage

Initialize IMU

```
imu_data_t imu_data;
imu_sensor_init(&imu_data);   /* zeroes the structure */
```

Poll IMU Data

```
result_t imu_result = poll_imu_sensor(&imu_data);

if (imu_result == RESULT_OK) {
    /* Acceleration (m/s²) */
    float accel_x = imu_data.accel[0];
    float accel_y = imu_data.accel[1];
    float accel_z = imu_data.accel[2];

    /* Angular velocity (°/s) */
    float gyro_x = imu_data.gyro[0];
    float gyro_y = imu_data.gyro[1];
```

```
    float gyro_z = imu_data.gyro[2];

    /* Magnetic field (Xsens arbitrary units) */
    float mag_x = imu_data.mag[0];
    float mag_y = imu_data.mag[1];
    float mag_z = imu_data.mag[2];
}
/* RESULT_ERR_COMMS: device not responding, or no fresh sample ready yet */
```

Advanced Functions

Update with Raw Values

```
result_t imu_sensor_update(
    imu_data_t *imu,
    float ax, float ay, float az,  /* Accelerometer values */
    float gx, float gy, float gz,  /* Gyroscope values */
    float mx, float my, float mz,  /* Magnetometer values */
    uint32_t timestamp
);
```

Calculate Acceleration Magnitude

```
float acceleration_magnitude = imu_get_acceleration_magnitude(&imu_data);
/* |a| = sqrt(ax² + ay² + az²), useful for impact and free-fall detection */
```

Orientation Helpers (from the gravity vector)

```
float pitch_deg = imu_get_pitch(&imu_data); /* atan2(ay, sqrt(ax²+az²)) in degrees */
float roll_deg  = imu_get_roll(&imu_data);  /* atan2(ax, sqrt(ay²+az²)) in degrees */
/* Assumes the device is relatively stationary */
```

Gyroscope Drift Check

```
/* true when all gyro axes are below the threshold (device at rest) */
bool stable = imu_check_gyroscope_drift(&imu_data, 1.0f);
```

Copying Data for Another Context

```
imu_data_t imu_copy;
imu_sensor_read(&imu_data, &imu_copy);
/* Plain struct copy, no locking: safe only if the source is not
 * being updated concurrently */
```

Sensor Ranges Used by the Driver Validators

Measurement	Accepted Range	Units
Acceleration	±16 g (±156.9 m/s²)	m/s²
Angular Velocity	±2000	°/s
Magnetic Field	±4900	driver limit (see unit note above)

Conversion Reference

From	To	Factor
g	m/s²	× 9.80665
rad/s	°/s	× 180/π (applied inside the driver)
Gauss	µT	× 100

Validation Functions

```
/* Driver-level validators (used by the main loop) */
bool imu_validate_accelerometer_range(imu_data_t *imu);  /* ±16 g in m/s² */
bool imu_validate_gyroscope_range(imu_data_t *imu);      /* ±2000 °/s */
bool imu_validate_magnetometer_range(imu_data_t *imu);   /* ±4900 */

/* Utility-library validators (sensor_basics.h) */
result_t validate_accelerometer_value(float accel_value); /* ±160 m/s² */
result_t validate_imu_data(float accel_x, float accel_y, float accel_z);
```

Protobuf Message Format

```
message SensorBoardIMUInfo {
    float accel_x;
    float accel_y;
    float accel_z;
    float gyro_x;
    float gyro_y;
    float gyro_z;
    float mag_x;
    float mag_y;
    float mag_z;
    SensorState state;
    IMUErrorCode error_code;
}

enum IMUErrorCode {
    IMU_NO_ERROR = 0;
    IMU_COMMUNICATION_FAILURE = 1;
    IMU_ACCELEROMETER_ERROR = 2;
    IMU_GYROSCOPE_ERROR = 3;
    IMU_MAGNETOMETER_ERROR = 4;
}
```

Common Applications

Impact Detection

```
float mag = imu_get_acceleration_magnitude(&imu_data);
if (mag > IMPACT_THRESHOLD) {
    /* High acceleration detected */
}
```

Tilt Detection

```
float pitch = imu_get_pitch(&imu_data);
float roll  = imu_get_roll(&imu_data);
```

Motion Classification

```
/* Static vs dynamic based on gyro magnitude */
float gyro_mag = sqrtf(gyro_x*gyro_x + gyro_y*gyro_y + gyro_z*gyro_z);
```

Integration Notes

- Single IMU instance in the main application (dual IMU planned)
- All nine axes (accel, gyro, mag) transmitted as independent fields at the main loop interval
- First poll performs one-time device configuration; a failing device degrades to IDLE / DISCONNECTED without blocking the loop
- On successful poll the main loop runs the three range validators and sets IMU_ACCELEROMETER_ERROR, IMU_GYROSCOPE_ERROR, or IMU_MAGNETOMETER_ERROR accordingly
- Timestamp tracking (HAL_GetTick) enables dead reckoning applications
- Filter algorithms can be applied to the raw data for smoothing
- Unit tested on host: init defaults, update/read round-trip, magnitude, pitch/roll, range validators (test/sensor_board/test_imu_sensor)

Load Cell

Load cells measure force/weight to detect object presence, evaluate structural loading, or monitor mechanical stress. The system supports a dual load cell configuration, each read through its own HX711 24-bit ADC using GPIO bit-banging. This is the most complete sensor driver on the board: it talks to real hardware with no compile gate.

Hardware Specifications

Parameter	Value
Sensor Count	2 (independent)
ADC	HX711 24-bit, one per load cell
Interface	GPIO bit-bang (DOUT input, SCK output)
Gain	Channel A, gain 128 (LOAD_CELL_GAIN_PULSES = 1)
Measurement	Force (Newtons) / Mass (grams) after calibration; raw counts always available
Ready timeout	200 ms (LOAD_CELL_READY_TIMEOUT_MS)
Supply	HX711 VCC 2.7 to 5.5 V, GND common with the STM32

Wiring and Pin Map (from firmware.ioc)

Unit	HX711 DOUT (data, low = ready)	HX711 SCK (clock)
0	PA5 (WEIGHT_INPUT_1)	PC7 (WEIGHT_CLOCK_1)
1	PA6 (WEIGHT_INPUT_2)	PB5 (WEIGHT_CLOCK_2)

Data Structure

```
typedef struct {
  int32_t raw_counts;              /* 24-bit two's complement reading */
  float force_newtons;
  float mass_grams;
  float scale_newtons_per_count;    /* default 1.0 (passthrough) */
  int32_t tare_offset_counts;
```

```
  bool is_calibrated;               /* true once load_cell_set_scale() called */
  bool read_ok;                     /* true if the last poll succeeded */

  /* HX711 hardware binding (set by load_cell_sensor_init_hw) */
  GPIO_TypeDef *dout_port;
  uint16_t dout_pin;
  GPIO_TypeDef *sck_port;
  uint16_t sck_pin;
  uint8_t gain_pulses;
} load_cell_data_t;
```

Initialization

Initialize Load Cells (as in main.c)

```
load_cell_data_t load_cell_data[2];

/* Unit 0: DOUT = PA5, SCK = PC7. Unit 1: DOUT = PA6, SCK = PB5. */
load_cell_sensor_init_hw(&load_cell_data[0], GPIOA, GPIO_PIN_5, GPIOC, GPIO_PIN_7);
load_cell_sensor_init_hw(&load_cell_data[1], GPIOA, GPIO_PIN_6, GPIOB, GPIO_PIN_5);
/* init_hw powers up the HX711 and auto-tares */

/* Alternative: load_cell_sensor_init(&data) zero-initialises WITHOUT a
 * hardware binding; poll() then returns RESULT_ERR_UNIMPLEMENTED */
```

Poll Load Cell Sensor

```
result_t lc_result = poll_load_cell_sensor(&load_cell_data[i]);
```

Data Access Functions

```
float force, scale;
float mass;
int32_t counts, tare;
bool valid;
```

```
load_cell_get_force_newtons(&cell, &force);
load_cell_get_mass_grams(&cell, &mass);
load_cell_get_raw_counts(&cell, &counts);
load_cell_get_calibration(&cell, &scale, &tare);
load_cell_sensor_is_valid(&cell, &valid);
```

Calibration Procedure

Two-Step Calibration

Step 1: Tare (Zero Load)

```
/* With nothing on the cell: average N raw reads and store as zero offset.
 * init_hw already does this automatically at startup. */
load_cell_tare(&cell, 10);
```

Step 2: Span (Known Weight)

```
/* Place a known mass, read raw counts, then:
 *   scale = known_force_newtons / (raw_counts - tare_offset_counts)   */
load_cell_set_scale(&cell, newtons_per_count);   /* sets is_calibrated = true */
```

Measurement Formulas

```
force_newtons = (raw_counts - tare_offset_counts) × scale_newtons_per_count
mass_grams    = force_newtons / 9.81 × 1000
```

Note: before calibration the default scale is 1.0 (passthrough): raw_counts is trustworthy but force_newtons and mass_grams are not physical units yet.

Protobuf Message Format

```
message SensorBoardLoadCellInfo {
    uint32 sensor_index;            /* 0 or 1 */
    float force_newtons;
```

```
    float mass_grams;
    int32 raw_counts;
    float scale_newtons_per_count;
    int32 tare_offset_counts;
    bool is_calibrated;
    SensorState state;
    LoadCellErrorCode error_code;   /* NO_ERROR, COMMUNICATION_FAILURE, INVALID_DATA */
}
```

Unit Conversions

From	To	Factor
Newtons	kilograms-force (kgf)	÷ 9.81
Newtons	pounds-force (lbf)	÷ 4.448
grams	kilograms	÷ 1000

Integration Notes

- Two independent units polled every main loop iteration; each is transmitted in its own envelope with its sensor_index
- Each sensor maintains separate calibration (tare + scale) and independent error reporting
- HX711 gain/channel is selected by extra SCK pulses after the 24 data bits (1 = channel A gain 128, 2 = channel B gain 32, 3 = channel A gain 64)
- The blocking wait for data-ready can take up to 200 ms per cell per poll; keep this in mind when reducing the loop interval
- A failed read maps to state ERROR with LOAD_CELL_COMMUNICATION_FAILURE; a read with implausible data maps to LOAD_CELL_INVALID_DATA

Pressure Sensor

The pressure sensors are analog force sensing resistor (FSR) pads read via ADC, intended primarily for robotic gripper force feedback: grip force sensing, object presence detection, and load distribution across two gripper pads. The system supports a dual sensor configuration.

Hardware Specifications

Parameter	Value
Sensor Count	2 (independent)
Interface	Analog ADC (compile gated, see Hardware Status below)
Board pins	PD15 (FORCE_ANALOG_DATA_1), PF3 (FORCE_ANALOG_DATA_2)
Output unit	kPa via linear conversion, plus raw voltage and temperature field

Hardware Status

The ADC path is compile gated by `PRESSURE_USE_ADC` because no ADC is enabled in CubeMX yet. Without the flag, `poll_pressure_sensor()` returns RESULT_ERR_UNIMPLEMENTED and both sensors report IDLE / DISCONNECTED (the firmware still links). To enable:

1. In CubeMX enable an ADC and the channel(s) for the force pins. On the STM32H753, PD15 has NO ADC function and PF3 = ADC3_INP5, so FORCE_ANALOG_DATA_1 must be moved to an ADC-capable pin.
2. Build with `-D PRESSURE_USE_ADC`.
3. Bind each unit with pressure_sensor_init_hw().

Conversion Model

```
voltage      = raw / adc_max × reference_voltage
pressure_kpa = voltage × scale_kpa_per_volt + offset_kpa
```

Defaults: scale_kpa_per_volt = 1.0, offset_kpa = 0.0 (passthrough until calibrated).

Data Structure

```
typedef struct {
  float pressure_kpa;
  float temperature_c;
  float voltage;
  bool is_calibrated;
  bool read_ok;              /* true if the last poll read succeeded */

  /* ADC binding (set by pressure_sensor_init_hw) */
  void *adc_handle;          /* ADC_HandleTypeDef* (void* keeps header HAL-free) */
  uint32_t adc_channel;      /* ADC_CHANNEL_x */
  uint32_t adc_max;          /* full-scale count (e.g. 65535 for 16-bit) */
  float reference_voltage;   /* ADC Vref+ in volts */
  float scale_kpa_per_volt;  /* linear gain (default 1.0) */
  float offset_kpa;          /* linear offset (default 0.0) */
} pressure_sensor_data_t;
```

Initialization

Initialize Pressure Sensors (as in main.c)

```
pressure_sensor_data_t pressure_data[2];
for (size_t i = 0; i < 2; i++) {
    pressure_sensor_init(&pressure_data[i]);
}

/* Once an ADC exists, bind it per unit: */
pressure_sensor_init_hw(&pressure_data[1], &hadc3, ADC_CHANNEL_5,
                        65535U, 3.3f);
```

Poll Pressure Sensor

```
result_t pr_result = poll_pressure_sensor(&pressure_data[i]);
```

Data Access Functions

```
float kpa, temp_c, voltage;
bool valid;

pressure_sensor_get_pressure_kpa(&sensor, &kpa);
pressure_sensor_get_temperature_c(&sensor, &temp_c);
pressure_sensor_get_voltage(&sensor, &voltage);
pressure_sensor_is_valid(&sensor, &valid);
```

Calibration

```
/* kPa = V × scale + offset; marks the sensor calibrated */
pressure_sensor_set_calibration(&sensor, scale_kpa_per_volt, offset_kpa);
```

Pressure Unit Conversions

From	To	Multiply By
bar	kPa	100
psi	kPa	6.895
atm	kPa	101.325
kPa	bar	0.01
kPa	psi	0.145
kPa	atm	0.00987

Function-based conversions (bar_to_psi, psi_to_bar) are declared in the utility library but currently commented out; see Sensor Board Utility Library.

Protobuf Message Format

```
message SensorBoardPressureInfo {
    uint32 sensor_index;            /* 0 or 1 */
    float pressure_kpa;
```

```
    float temperature_c;
    float voltage;
    bool is_calibrated;
    SensorState state;
    PressureErrorCode error_code;   /* NO_ERROR, COMMUNICATION_FAILURE, INVALID_DATA */
}
```

Applications

Robotic Gripper Control (Primary Use Case)

- Grip force feedback for object handling
- Object presence detection (pressure spike threshold)
- Adaptive compliance for varying object sizes and materials
- Dual sensors support load sharing across gripper pads

Possible Secondary Uses (not implemented)

- Depth sensing (water), altitude sensing (air), system pressure monitoring, if a suitable transducer replaces the FSR pads

Integration Notes

- Two independent units polled every main loop iteration, each transmitted in its own envelope with its sensor_index (logged under the name "Force0"/"Force1")
- Each sensor maintains independent calibration and error reporting
- The temperature_c field exists for future compensation algorithms; no temperature source is wired up yet
- Until the ADC is enabled the sensors are harmless placeholders: IDLE / DISCONNECTED, zeroed values

Testing

Test organization, the Unity testing framework usage, coverage per suite, manual hardware testing checklists and a debugging/troubleshooting guide. Reminder: the sensor board code was only partially tested; unit tests cover the pure-logic parts of the drivers, and end to end hardware validation was not completed by the 2025-2026 team.

Test Organization

Suite	Location
test_sensor_basics	test/sensor_board/test_sensor_basics/test_sensor_basics.c
test_ph_sensor	test/sensor_board/test_ph_sensor/test_ph_sensor.c
test_imu_sensor	test/sensor_board/test_imu_sensor/test_imu_sensor.c

Test Framework

- **Framework:** Unity (open source C testing framework)
- **Build system:** PlatformIO (env:sensor_board, test_filter = sensor_board/*)
- **Test type:** driver logic tests exercising the data structures and math directly; the hardware access paths (ADC, HX711 GPIO, EXTI) are not mocked and not covered

Building and Running Tests

```
# All sensor board tests
pio test -e sensor_board

# One suite
pio test -e sensor_board -f test_ph_sensor

# Verbose output
pio test -e sensor_board -f test_sensor_basics -v
```

Coverage Per Suite

test_sensor_basics

- Accelerometer boundary values: ±160.0 accepted, ±160.1 rejected
- Multi-axis validation: one bad axis fails validate_imu_data()
- Temperature and pressure conversion tests exist but are commented out together with their implementations

test_ph_sensor

- Initialization defaults (raw 0, voltage 0, pH 7.0, stored reference voltage)
- ADC to voltage to pH conversion with a 12-bit ADC scale
- Clamping at the extremes (ADC 0 clamps to pH 14, full scale clamps to pH 0 with the test calibration)
- Calibration changes the measurement (offset and slope applied)

test_imu_sensor

- Initialization zeroes all axes and the timestamp
- Update and read round-trip for accel/gyro/mag and timestamp
- Acceleration magnitude (3-4-12 triangle gives 13)
- Pitch and roll helpers from accelerometer data
- Range validators accept zeros and reject out-of-range values

Manual Hardware Testing Checklist

1. Flash with `pio run -e sensor_board -t upload` and open the serial monitor at 115200 baud
2. Confirm the boot banner and each "init completed" line (IMU, pH, load cells, pressure, flow, pump, Ethernet)
3. Confirm the three LEDs toggle every 5 seconds (loop heartbeat)
4. Check the per-sensor status lines: connected hardware should read OPERATING | OK, absent hardware IDLE | DISCONNECTED
5. Load cells: press on each cell and watch raw_counts/force change; verify tare at startup reads near zero
6. Flow and pump: with tubing wet, enabling the pump must produce flow pulses; "commanded on but no flow detected" indicates a dry/absent pump or a not-configured EXTI4 line
7. Network: set sendUDP = true, then capture UDP datagrams on port 1500 at 192.168.0.222 and decode with the PBEnvelope schema

8. Send a SensorBoardPumpInfo command packet and verify the pump speed changes

Debugging & Troubleshooting

Issue	Cause	Solution
Sensor IDLE / DISCONNECTED	Not connected, or driver compile gated	Check wiring; for pH/pressure verify the PH_SENSOR_USE_ADC / PRESSURE_USE_ADC build flags and the CubeMX ADC config
Sensor ERROR	Communication failure	Verify HX711 wiring and timing, I2C address and pull-ups, ADC channel binding
Invalid data	Out of range values	Check calibration parameters (pH slope/offset, load cell scale/tare, pressure scale/offset)
Flow always 0	EXTI4 not enabled in CubeMX	Configure PA4 as EXTI4 rising edge and enable the EXTI4 NVIC line
Pump OPERATING / DISCONNECTED	No flow while commanded on	Pump absent, dry, or stalled; or the flow sensor is not installed/configured
No UDP packets	Transmit gate or addressing	Set sendUDP = true; check IP/MAC constants, MAC filtering, and that port 1500 is not blocked
Low heap warning	Memory leak or queue growth	Review protobuf encode/free paths and UDP queue sizes
Serial monitor silent	Wrong port or baud	Check the ST-Link COM port and 115200 baud; verify LOG_init ran

Reference

Source code references, build configuration, hardware datasheet pointers, useful commands and the pre-deployment checklist. If you made it till here, you a true G.☮

Source Code References

Main Application

File	Purpose
src/sensor_board/main.c	Main entry point, MainTask, sensor loop, packet handlers

Sensor and Actuator Drivers

Component	Location
IMU	components/sensor_board/imu/imu_sensor.h / .c
pH	components/sensor_board/ph/ph_sensor.h / .c
Load Cell (HX711)	components/sensor_board/load_cell/load_cell_sensor.h / .c
Pressure (FSR)	components/sensor_board/pressure/pressure_sensor.h / .c
Flow Sensor	components/sensor_board/sampling/flow_sensor/flow_sensor.h / .c
Pump	components/sensor_board/sampling/pump/pump.h / .c
Utilities	components/sensor_board/sensor_basics/sensor_basics.h / .c

Shared Components

Component	Location
Networking (LwIP glue, UDP)	components/common/networking/
Network addresses and port	components/common/networking_constants/ip_mac_constants.h
Packet dispatcher	components/common/packet_dispatcher/
Protobuf encode/decode helpers	components/common/pb_message/

Result codes and TRY macro	components/common/result/
Logging	components/common/logging/

Protobuf Definitions

Message definitions (PBEnvelope, SensorBoardPHInfo, SensorBoardIMUInfo, SensorBoardLoadCellInfo, SensorBoardPressureInfo, SensorBoardFlowSensorInfo, SensorBoardPumpInfo, SensorBoardDiagnostics) live in the **ERC-Protobufs** git submodule and are compiled to C by nanopb during the PlatformIO build. If the build cannot find the .pb.h headers, initialize the submodule:

```
git submodule update --init ERC-Protobufs
```

Build Configuration

File	Purpose
platformio.ini	Build configuration for all boards (env:sensor_board for this one)
components/sensor_board/firmware/firmware.ioc	CubeMX device configuration
components/sensor_board/STM32H753XX_FLASH.ld	Linker script

Development Workflow

Useful Commands (PlatformIO CLI)

Check this out for compiling code and more about project structure- Project Structure

```
# Build the sensor board firmware
pio run -e sensor_board

# Flash to the Nucleo board
pio run -e sensor_board -t upload

# Serial monitor (115200 baud)
pio device monitor -b 115200
```

```
# Run the unit tests
pio test -e sensor_board
```

Development Cycle

1. Change peripherals in CubeMX (firmware.ioc), regenerate with Keep User Code
2. Implement or update the driver in components/sensor_board/
3. Build and run unit tests on the host
4. Flash, watch the serial log, verify the per-sensor status lines
5. Enable sendUDP and verify packets on the network

Hardware References

Device	Model	Protocol	Note
Microcontroller	STM32H753ZI (NUCLEO-H753ZI)	n/a	ARM Cortex-M7, 480 MHz capable (running at 64 MHz), 2 MB Flash; ST STM32H7 reference manual
Ethernet PHY	LAN8742	RMII	10/100 Mbps auto-negotiation
pH	DFRobot SEN0161	Analog ADC	40-sample averaging, 5 V board (scale output below 3.3 V)
IMU	Xsens Avior	I2C1 (PB8/PB9), Xbus	Address 0x6B, 100 Hz; driver untested on hardware (mtidocs.xsens.com)
Load Cell ADC	HX711 (×2)	GPIO bit-bang	24-bit, channel A gain 128
Pressure	FSR pads (×2)	Analog ADC	Compile gated, needs ADC in CubeMX
Flow	FM-PS2216	GPIO EXTI pulses	40 to 150 ml/min, 5.5 pulses/ml
Pump	Grothen 12 V DC mini peristaltic	PWM (TIM3 CH3)	Single MOSFET, unidirectional, open loop

Network Quick Reference

Item	Value

Board IP	192.168.0.111 (static, no DHCP)
Destination (sample board)	192.168.0.222
UDP port	1500
Netmask / Gateway	255.255.255.0 / 192.168.0.1

Quick Reference Checklist

Before Deployment

- ERC-Protobufs submodule initialized, firmware builds clean
- All connected sensors responding (OPERATING | OK in the log)
- Network IP/MAC configured and sendUDP enabled
- Calibration set for pH (offset/slope) and load cells (tare/scale)
- Serial monitor showing sensor data at 115200 baud
- Heap usage healthy (well above the 4096 byte critical threshold)
- UDP packets reaching 192.168.0.222:1500 and decoding as PBEnvelope

Monitoring in Production

- Watch per-sensor state/status codes in the log lines
- Monitor the free heap trend printed each loop
- Verify data ranges match expectations (pH 0-14, flow below 150 ml/min)
- Track error rates per sensor and pump/flow cross-check warnings

Hope you had fun.☮ End of documentation for the Sensor Board.

May The Force Be With You or Live Long and Prosper, depending on what you like.... but remember the Dark Side always has cooler toys

The Sensor Board is the coolest board ~ Mybrosky