ChatGPT

Comprehensive DIY Ocean-Surface Motion Sensing Buoy: Feasibility, Design, and Implementation

Overview

Designing a do-it-yourself ocean surface motion sensing buoy is a challenging but achievable project for an electronics hobbyist. The goal is to build a low-cost, robust buoy equipped with a 3-axis gyroscope/accelerometer (IMU) and supporting hardware to **measure and record fine-grained wave motion** in X, Y, Z axes. Such a device should operate continuously for 3–4 weeks in open coastal waters, logging or transmitting multiple samples per second (targeting ~100 Hz sampling, i.e. <10 ms intervals, for near-continuous motion capture). Key design considerations include **hardware selection** (sensors, microcontroller, power), **data handling** (local storage vs. wireless transmission), **waterproofing and buoy construction**, **sampling/calibration strategies**, and **deployment logistics**. This report surveys comparable DIY projects and research, examines component options, and proposes two example build plans (a basic logger vs. an enhanced solar-powered transmitter), including a bill of materials and practical implementation notes. By leveraging inexpensive sensors and microcontrollers, hobbyists can indeed achieve scientific-quality wave measurements at a fraction of the cost of commercial instruments [1] [2], provided careful attention is paid to power management, sensor calibration, and marine durability.

Comparable DIY Buoy Projects

Several open-source and low-budget projects serve as valuable references for this undertaking:

- **T3chFlicks "Smart Buoy" (2019)** – A multi-sensor coastal buoy that measures wave height, period, wave energy, temperature, pressure, etc., using an Arduino Nano and a GY-86 IMU (MPU6050 accel/gyro + magnetometer + barometer) [3] [4]. It transmits data via radio to a shore-based Raspberry Pi dashboard, with power from an 18650 Li-ion battery recharged by small solar panels [5] [6]. The buoy's 3D-printed spherical enclosure is coated in epoxy for waterproofing [7] [8]. Notably, the team derived wave height from barometric pressure changes (as a proxy for altitude) and wave period via zero-crossing analysis, achieving a crude but effective wave characterization [9] [10]. Wave direction proved difficult, highlighting an ongoing challenge even in research [11] [12].

- **Open Source Ocean Data Buoy (Nick Raymond, 2014–2018)** – A documented project building a wave and weather buoy with modular electronics. Early versions used an Arduino (Atmega328P @ 3.3 V) with an **Adafruit BNO055 9-DOF IMU**, GPS for location/time, an SD card for logging, and an XBee or LoRa radio for telemetry [13]. The BNO055 provided accelerometer, gyroscope, and magnetometer data with onboard sensor-fusion for orientation. Power was supplied by a pack of Li-Po batteries (e.g. 3× 2200 mAh in parallel) with plans for solar charging [14] [15]. This project encountered typical challenges: limited code space on the Arduino (32 KB) for all sensor libraries [16] (leading to considerations of using dual microcontrollers or a more powerful board), and the need to run everything at low power (3.3 V, 8 MHz) for long deployments [17]. Nevertheless, it demonstrated

a feasible architecture: a Pro Trinket (Atmega328) controlling a BNO055 IMU, logging to SD, and intermittently transmitting data via radio.

- **"Compact Low-Cost Wave Buoy" (Yurovsky & Dulov, 2018)** – A research prototype developed by oceanographers, showing that hobby-grade IMUs can yield scientific wave data [1] [18]. Their 60 g buoy used an MPU-9250 9DOF sensor and Arduino Pro Mini (3.3 V/8 MHz) to log raw acceleration, rotation, and magnetic data at ~160 Hz to a 4 GB microSD card [19] [20]. Powered by a tiny 300 mAh Li-ion battery, it ran ~10 hours continuously [21] (sufficient for short deployments). The electronics were housed in a 10 cm long, 2.5 cm diameter waterproof capsule (a sealed PET plastic bottle preform) inserted in a foam disk for buoyancy [22]. **Post-processing** on retrieved data computed wave spectra and parameters, confirming that the low-cost IMU's noise was negligible for common wave frequencies [18]. This project intentionally sacrificed endurance for simplicity, but it provides a baseline: cheap MEMS sensors can indeed capture wave motions reliably, and a **small buoy** can measure shorter wavelength "chop" that larger buoys filter out [23].

- **OpenWave (WattNotions, 2019)** – A blog-documented effort to develop an open-source wave sensor package [24] [2]. The author reviewed low-cost IMUs and chose the Bosch BNO055 (with built-in ARM Cortex M0 for sensor fusion) to avoid writing complex sensor-fusion algorithms [25] [26]. The BNO055 can directly output orientation (Euler angles or quaternion) and **linear acceleration (acceleration minus gravity)** at 100 Hz [27] [28], simplifying the calculation of wave motion. Initial tests (using a rotating arm to simulate waves) showed promising correlation between computed wave heights and actual motion, though dealing with integration drift required careful filtering and segmentation of the data (e.g. resetting integration at wave zero-crossings) [29] [30]. OpenWave underscores the benefit of using advanced IMUs like BNO055 or similar (e.g. BNO085) to get high-level motion data with minimal coding.

These projects illustrate different approaches and trade-offs. The Smart Buoy and Open Source Buoy aimed for longer-term deployment with solar power and wireless data links, whereas the research and OpenWave prototypes focused on short-term logging and post-analysis. Together, they inform the **feasibility** of a DIY wave buoy and guide our design choices.

Hardware Components and Options

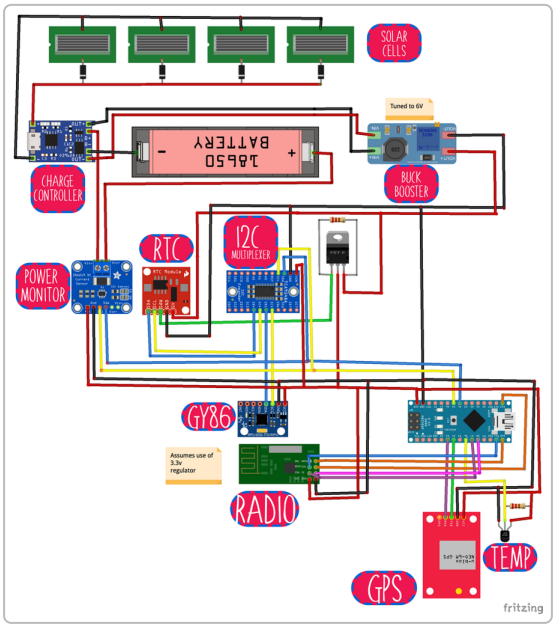

Figure: Example system schematic from the Smart Buoy project (T3chFlicks), illustrating the integration of sensors (GY-86 IMU, temperature), microcontroller (Arduino Nano), radio (NRF24/LoRa), GPS, real-time clock, power management (18650 Li-ion battery, charge controller, solar panels), and other modules [31] [5].

A robust motion-sensing buoy requires a combination of sensor, processing, power, and communication hardware. Key components and options are discussed below:

IMU Sensor (Gyroscope/Accelerometer) Options

The core of the system is the **inertial measurement unit (IMU)** that captures 3-axis acceleration and angular velocity, and possibly magnetic heading. Several affordable IMU modules are suitable:

- **MPU-6050 (6-DOF)** – A widely used 3-axis accelerometer + 3-axis gyroscope. It's low-cost (~$5) and can sample at high rates (up to 1 kHz internally), but it provides raw data only. An external processor must handle drift correction and, if needed, combine with an external magnetometer for orientation. The MPU-6050 includes a Digital Motion Processor (DMP) that can do limited sensor fusion (accel + gyro) internally, outputting a filtered orientation estimate, but using the DMP requires specific libraries and does not incorporate magnetometer data [32].

- **MPU-9250 / MPU-9255 (9-DOF)** – This IMU adds a 3-axis magnetometer to the accelerometer/gyro package, enabling full 9-DOF sensing. It has similar cost and capabilities to the MPU-6050 plus an extra magnetometer channel. Like the 6050, it has a DMP that fuses accel and gyro data, but magnetometer fusion must be done externally [32]. The **noise performance** of MPU-92xx is generally good for wave measurements – the research buoy showed that its MEMS noise and drift did not significantly affect common wave signals [18]. Using all 9 axes allows post-processing to reconstruct the buoy's orientation and vertical displacement by combining acceleration and tilt information [33] [34], albeit with substantial algorithm complexity.

- **Bosch BNO055 (9-DOF with onboard fusion)** – A more advanced (but costlier, ~$30) IMU that integrates an ARM microprocessor performing real-time sensor fusion. The BNO055 outputs orientation (Euler angles or quaternions) at up to 100 Hz, as well as calibrated acceleration, angular velocity, and even linear acceleration (acceleration minus gravity) [27] [28]. This greatly simplifies software – the buoy can directly read tilt-compensated motion and perhaps integrate vertical acceleration to get wave elevation. The trade-off is power consumption (BNO055 draws ~12 mA) and some reported long-term drift issues (the internal calibration needs to be maintained, and extreme temperatures may throw it off). Nonetheless, the BNO055 (or its newer variant BNO085/BNO080) can save significant development time by providing "ready-to-use" motion data [35].

- **Alternative IMUs** – Other modern IMUs like the **ICM-20948** (InvenSense) or **LSM9DS1** (STMicro) offer 9-DOF sensing. They lack built-in fusion but are inexpensive and have lower power consumption. For example, the ICM-20948 is an update to the MPU-9250 with similar capabilities. If one is comfortable implementing sensor fusion or only needs raw data for offline processing, these chips are viable. There are also 6-DOF accelerometer/gyro units with very low drift (e.g. Analog Devices ADXL and ADXRS series), but their cost is higher.

Marine performance considerations: In a buoy, the sensor will experience slow tilts and accelerations from waves (typically 0.1–1 Hz motion) as well as possibly short high-frequency jostling (from chop or wind). All the listed IMUs can easily sample faster than these dynamics (100 Hz or more). Temperature stability is important: the enclosure can heat up in sunlight, which can cause sensor bias shifts. The IMU's internal temperature sensor can be used to calibrate any drift if characterized. Magnetic interference should be minimized – if using a magnetometer, keep it away from batteries or large metal objects in the buoy, and perform a calibration "spin" in a disturbance-free area before deployment. Overall, a **9-DOF IMU** is recommended for the fullest picture (allowing orientation tracking and drift correction), but if simplicity and cost are paramount, a 6-DOF sensor with careful calibration can suffice for measuring wave *motion* (acceleration/tilt). The addition of a **pressure sensor** is also very useful: a barometric pressure sensor (like MS5611 or BMP388) can act as an altimeter to directly measure the buoy's vertical displacement due to waves [9], which is an effective way to get wave height without double-integrating accelerometer data.

Microcontroller and Data Logging

At the heart of the buoy, a microcontroller (MCU) is needed to read the sensors, timestamp the data, and either store it or transmit it. Key considerations are **power efficiency**, **processing capability**, and **memory/storage interfaces**:

- **Arduino (8-bit AVR) Boards:** Classic Arduino boards (Nano, Uno, Pro Mini) use the ATmega328 microcontroller. They have enough I/O and ADC for sensor interfacing and can be run at 3.3 V/8 MHz for low power. Their 32 KB program memory and 2 KB RAM, however, limit complex logic or buffering. Logging raw IMU data at 100 Hz to an SD card *is* possible (as shown by Yurovsky et al. [19] [20]), but adding wireless communication, GPS, and multiple sensors can hit memory limits quickly [16]. Some projects addressed this by using multiple microcontrollers or moving to a more powerful board. If using an Arduino Pro Mini or Nano, it's wise to use efficient libraries and possibly avoid heavy frameworks; the Smart Buoy team even removed their SD card module because the file-system library consumed too much memory on the Nano [36]. A related option is Arduino Mega (128 KB flash, 8 KB RAM), but it's larger and uses more power, and a 3.3 V variant is needed to interface directly with IMUs.

- **Modern 32-bit Microcontrollers:** Boards like the *Arduino Zero* or *Adafruit Feather M0 (SAMD21)*, the *Teensy* series, or the *ESP32* provide far more processing power and memory. For example, an ESP32 has 240 MHz dual-core CPU, ~520 KB RAM, and built-in Wi-Fi/Bluetooth, at low cost (~$10). It can easily handle logging and transmitting data, and can run in deep sleep between samples if needed. However, running an ESP32 continuously at full power can drain a battery quickly (it draws ~80 mA active). The Raspberry Pi Pico (RP2040 microcontroller) is another attractive option: dual-core, plenty of RAM, and it can run MicroPython or C++ code; it has deep-sleep modes and is very low-cost (~$4). Unlike a single-purpose Arduino, these 32-bit MCUs require a bit more setup in code, but they enable advanced functionality (e.g. local data buffering, on-device signal processing) that an AVR might struggle with. For a 3–4 week deployment on battery, ensure the MCU can enter a low-power sleep state when idle (and wake via a timer/RTC interrupt to sample). Many microcontroller-based boards (Feathers, etc.) have libraries for managing sleep and RTC timers.

- **Single-Board Computers (e.g. Raspberry Pi):** One might consider using a Raspberry Pi Zero or similar Linux SBC on the buoy. While a Pi Zero can interface with sensors and even directly upload data via cell network or Wi-Fi, it's **not ideal for low-power** operation – it typically draws 100–200 mA even idle. The boot-up and shutdown are also less robust in a sudden power-loss scenario (risk of SD card corruption). Therefore, pure microcontroller solutions are typically preferred for long-term buoys. A Raspberry Pi *is* great for the **base station** on shore (like receiving radio data and running a web dashboard [37] [38]), but on the buoy itself a Pi would necessitate a much larger power supply.

- **Storage (Data Logging):** If transmitting data in real-time is not guaranteed (or to have a backup), local storage is needed. The common approach is using a **microSD card** module, which allows gigabytes of data logging (necessary for high sample rates over weeks). Writing to SD in a power-efficient way involves buffering: writing 512 byte blocks at intervals rather than byte-by-byte. The Arduino SD library can be memory-heavy [36], but alternatives like SdFat or using a lower-level SPI approach can help. Another storage option is **FRAM (Ferroelectric RAM)** or **Flash** chips – these can log data without the complexity of a file system, but they usually store only a few megabytes. For instance, a 4 MBit FRAM (0.5 MB) would be filled in a couple of minutes at 100 Hz logging, so not suitable alone. Thus, an SD card (8–16 GB, class 10) is the practical choice; just be sure to handle file closes and power-off properly to avoid corruption. Use a real-time clock or the MCU's millisecond timer to timestamp each record so data can be aligned to actual time later.

- **Real-Time Clock (RTC):** For multi-week deployments, the MCU's internal clock may drift; adding a cheap RTC module (e.g. DS3231) provides accurate timekeeping with a backup coin-cell. The RTC can also serve a dual purpose: some designs use the RTC alarm as a wake-up trigger to power the system on at intervals [5]. For example, the Smart Buoy kept the RTC powered continuously and had it control a FET to connect the battery to the rest of the circuit on a schedule [39]. This allowed the entire sensor and MCU system to be truly off between measurements, drastically saving power. In our design, we can similarly use an RTC to either interrupt the MCU from sleep or physically switch a power latch circuit.

Power Supply and Battery

Power is one of the most critical aspects. We need a **power source** that can sustain the buoy for ~3–4 weeks (~**20–30 days**) continuous operation, and a strategy to recharge or conserve energy. Key components:

- **Battery Choice:** Common options are lithium-ion rechargeable cells (e.g. 18650 Li-ion) or lithium-iron-phosphate (LiFePO4) cells. A single 18650 (Li-ion) has ~3.6 V nominal, 3000 mAh capacity (≈11 Wh). Using one directly with a 3.3 V regulator is convenient, and many microcontrollers and sensors can run off 3.3 V. LiFePO4 cells have a slightly lower nominal voltage (3.2 V) but remain above 3.0 V for most of their discharge, meaning some 3.3 V devices can be powered directly without a regulator (or a DC-DC booster can maintain a stable 3.3 V). LiFePO4 are more stable and have longer cycle life (good for many recharge cycles if using solar) and are safer to leave fully charged. For a ~1 month deployment, the required capacity depends on the average load:
- If the system averages e.g. 5 mA (possible with aggressive sleep duty cycling), 5 mA × 24 h × 30 ≈ 3600 mAh, so one 18650 could suffice.
- If it averages 50 mA (more continuous sensing/transmitting), that's 36,000 mAh needed (which would require a parallel pack of many cells or daily recharging via solar).

Clearly, **power management** is crucial. A realistic approach is to duty-cycle the system: for example, sample at 100 Hz but in bursts or sleep the radio when not in use. The Smart Buoy dynamically adjusted its sleep interval based on battery voltage (if sunlight was low, it slept longer) [40] [36]. For our design, a **hybrid strategy** could be: - Minimal design: use a high-capacity battery and log data locally, letting the MCU sleep between sensor readings if possible. - Enhanced design: use solar panels to extend battery life indefinitely and/or allow more frequent data transmission.

- **Solar Panels and Charging:** Adding small solar panels can greatly extend deployment. For example, four mini-panels (~5 V, 0.5 W each) were embedded on the Smart Buoy's surface [41], feeding a charge controller (often a Li-ion charging IC with MPPT or a basic solar charger module) [41] [42]. A suitable module is the Adafruit Solar LiPo charger or generic MPPT charger, which prevents overcharging and manages the panel's operating point. **Blocking diodes** are used to prevent reverse current at night [41]. With solar, the buoy can theoretically operate indefinitely, but one must account for short winter days or cloudy periods – the battery should have enough capacity to last a few days without sun. **LiFePO4** cells pair well with solar because they can be held at full charge safely and tolerate many cycles; a 3.2 V LiFePO4 can also directly power 3.3 V logic (some designs use 4 cells (12.8 V) with a regulator, but for simplicity a single cell plus a boost converter to 5 V or 3.3 V is fine).

- **Low-Power Operation:** Use efficient regulators (a low-dropout regulator or buck converter for any voltage conversion). Avoid linear regulators that waste power as heat; if the system runs off a single-cell Li-ion (~4.2 V down to 3.5 V) and only needs 3.3 V, a low-dropout linear reg is okay. If using 5 V logic or sensors, a buck converter is better. Also consider powering down sensor modules when not needed: e.g. some IMUs have sleep modes or can be turned off via a MOSFET on their Vcc. The radio or GPS can likewise be power-gated. The trade-off is complexity versus simplicity – a simple design might just run everything at 3.3 V continuously and rely on sleep modes, whereas an extreme low-power design turns off power-hungry components completely between readings.

- **Estimating Battery Life:** As an example, suppose our buoy uses an ESP32 logging to SD and sending via LoRa. Active current might be ~80 mA during a measurement/transmit cycle of 2

seconds, then the ESP32 sleeps for 10 seconds at 0.2 mA. The average would then be ~[80 mA*2* + *0.2 mA*10] / 12 ≈ 13.5 mA. Over 30 days, that's about 13.5*24*30 ≈ 9720 mAh. Two parallel 18650 (≈6000 mAh) would not quite suffice; solar charging a bit each day could top up the rest. In contrast, a pure logger with an Arduino Pro Mini waking briefly to sample could achieve <1 mA average draw [21], making a single 18650 more than enough. These estimates guide whether solar is necessary or just a nice addition.

Data Transmission vs. Local Storage

A crucial design decision is whether to **transmit data in real-time** (or near real-time) or store it locally for post-recovery analysis. Each approach has pros and cons:

- **Local Storage (Standalone Logger):** The buoy records all data to on-board memory (e.g. SD card). This maximizes data reliability (assuming the buoy is recovered) and bandwidth – you can log high-frequency raw data without concern for wireless throughput. It also simplifies power usage, as you don't need an always-on radio. Many wave research buoys use this method, performing analysis only after retrieval [43] [44]. **Downside:** if the buoy is lost at sea or leaks, the data is lost. Also, you don't get real-time insights. For a coastal deployment ("swimming distance" from shore), retrieving the buoy periodically (e.g. every few weeks) might be acceptable. One mitigation is to use a **redundant logger** – for example, write to two SD cards (some loggers do this in case one fails) or use an additional non-volatile memory for critical summary data.

- **Wireless Transmission:** Various wireless options can send data to shore, improving data accessibility and potentially allowing immediate alerts or analysis. Options include:

- **LoRa (Long Range Radio):** LoRa modules (868 MHz in Europe, 915 MHz US, or 433 MHz bands) offer long-range, low-bandwidth communication. With a simple whip antenna at the buoy and a receiver onshore, ranges of 1–2 km over water are achievable. LoRa is low-power when transmitting short packets infrequently. For example, a LoRa radio might draw ~50 mA for tens of milliseconds per packet. The limitation is data rate – only a few kbps at most. Sending every raw sample (100 Hz) is impractical, but you could transmit decimated data or summary statistics periodically (e.g. send wave height and period every minute, while storing full data on SD). LoRa also supports a simple acknowledge/retry, which can help ensure data packets are received. The Smart Buoy used an **NRF24L01+** radio (2.4 GHz) initially [45] – those are short-range (a few hundred meters) compared to LoRa, but very low cost. An upgraded LoRa-based approach would increase range. Another benefit: no subscription needed (unlike cellular).
- **Cellular (LTE/GPRS):** A GSM/GPRS modem (like SIM800) or LTE-M/NB-IoT module can send data via the mobile network. This is suitable if the deployment is within cell coverage (near coast likely yes). It allows data to be sent to a server or cloud in real-time. However, cellular modules consume significant power, especially when searching for signal or transmitting (hundreds of mA spikes). They also add cost (hardware $20+ and need a SIM card/data plan). For a hobby project, this might be overkill unless real-time remote access is crucial.
- **Wi-Fi:** If the buoy is very close to shore or a pier, Wi-Fi could be used to offload data when in range (for instance, connecting whenever the buoy comes within 100 m of a receiver, if it's drifting – but in our case the buoy is anchored and stationary, so Wi-Fi would need to reach it out in the water). Wi-Fi is power-hungry to keep on constantly, but one strategy could be to store data and activate a Wi-Fi module at set times to upload in bulk if a hotspot is reachable.

- **Satellite:** For completeness, satellite transmitters (like Iridium short burst data or Spot trackers) can send sensor data from anywhere. These are expensive (hundreds of dollars for hardware, plus per-message costs) and usually only used for remote open-ocean buoys. Likely not in scope for a DIY coastal buoy due to cost.

In summary, for a **near-shore deployment**, a **LoRa radio link** to a local base station is an attractive option: it's relatively low power and can cover the distance. The base station could be a Raspberry Pi or laptop with a LoRa USB dongle, receiving packets and maybe uploading them to a web service. This way, even if the buoy isn't recovered, you have the transmitted data up until loss. **However**, it is wise to also log data locally as a backup (the buoy can do both). The Smart Buoy did exactly this initially (Arduino + radio), and they only abandoned the SD card due to code size limits, not concept [36]. With a more capable MCU, dual logging is feasible. One must implement a robust buffering scheme so that transmission delays don't disrupt the sensor sampling.

To manage power, the buoy could transmit at a lower rate (for example, send a packet of summary stats every 10 seconds, and maybe a snippet of raw data occasionally), while writing full detail to SD. This reduces radio on-time. Also consider using a **GPS module** if drift or retrieval is a concern – a GPS can log the buoy's position (e.g. to ensure it stays anchored, and help find it). GPS will draw ~20 mA when active; one could schedule it to fix position say once an hour to save power. The Smart Buoy did include GPS [46] [47] for location logging.

Buoy Design, Waterproofing and Buoyancy

The physical design of the buoy must keep the electronics dry and the sensors functioning, while surviving ocean conditions (sun, salt, waves). Key elements:

- **Enclosure:** Many DIY buoy builders use off-the-shelf waterproof enclosures or make custom ones. Common solutions:
- PVC piping with end caps: A section of PVC or ABS pipe with solvent-welded or O-ring sealed end caps can make a strong waterproof cylinder. Through-hull fittings (cable glands) can be added for antennas or sensor probes. It's important to test the seals under pressure (e.g. submerge the closed tube in water for hours). PVC electrical junction boxes (with gasketed lids) are another option for a rectangular container.
- Waterproof cases: Small Pelican™ or OtterBox™ cases can be repurposed as buoy enclosures. They seal well and are rugged. One might need to add foam for buoyancy or mount them in a larger float so they stay upright.
- 3D-printed enclosure: The Smart Buoy used a **3D-printed sphere** (~20 cm diameter) in two halves [48]. The top had cutouts for solar panels and an antenna, the bottom had a hole for a temperature sensor and a point to attach the anchor line [48]. They printed in PETG plastic (which is more UV and water-resistant than PLA) and then **sanded, primed, and coated it in epoxy** to fully seal any micro-gaps [49]. This approach allows custom shapes (spherical for omni-directional wave response), but requires a good printer and post-processing. **Epoxy coating** is highly recommended for any 3D print in water to ensure it's watertight.

- Reused containers: The research buoy's use of a **PET bottle preform** (essentially a thick plastic tube with screw cap) is clever [22] – these preforms are cheap, extremely strong (designed to become pressurized bottles), and with an O-ring in the cap they seal well. The downside is limited internal volume (in their case 2.5 cm diameter tube).

- **Buoyancy and Stability:** The buoy must float and ideally keep the sensor oriented. To measure wave **surface motion**, it's typically best for the buoy to *follow* the water movement (i.e. be as small/light as possible so it moves with even small waves) [23]. However, if it's too small/light, it might get knocked around or flipped. A common design is a spherical or disk shape with **weight at the bottom** to maintain upright orientation (like a self-righting buoy). For example, you could place the battery low in the housing or add a ballast (a small metal weight) at the bottom of the buoy so that the center of mass is below the center of buoyancy. The Smart Buoy's spherical enclosure likely had the battery and electronics in the lower half, contributing to stability. The open-source buoy used a disc-shaped foam float around a central capsule [50], which increases buoyancy and stability on the water surface.

Ensure that the buoy has enough buoyant force to carry the weight of the electronics *plus* some safety margin. For instance, a 10 cm diameter sphere displaces about 0.5 kg of water; if your hardware weighs 200 g, that's okay. A larger buoy will ride more smoothly, but note it will **filter out** the very short wavelength waves (if the buoy diameter is, say, 0.5 m, it won't respond to 0.1 m ripples) [51] [52]. For "calm to choppy seas" (wave heights perhaps 0.1–1 m), a volleyball-sized or smaller buoy should capture the motion well.

- **Antenna and Sensor Placement:** If using wireless transmission, you'll need an antenna sticking out. A waterproof antenna (rubber duck antenna with a seal, or a wire antenna potted in epoxy) should be placed as high as possible on the buoy for range, but not so large that it upsets balance. Similarly, if you have a **water temperature probe**, it should protrude into the water but the penetration point must be sealed (e.g. use epoxy or marine sealant around the sensor wire – the Smart Buoy used hot glue and StixAll adhesive to seal cable penetrations [53]). Inertial sensors should be mounted firmly inside (use standoffs or foam pads to isolate high-frequency vibration). It's wise to conformally coat the sensor PCB to protect it from any condensation or salt that does get in.

- **Waterproofing Techniques:** Use **marine-grade sealants** or epoxy on any seams. Silicone caulk can work for temporary seals but tends to peel over time; two-part epoxy or polyurethane sealants (like 3M 5200) are more permanent. O-ring seals are excellent for lids/caps that may need to be opened; just keep the O-ring clean and greased. Provide a **dessicant pack** inside the enclosure to absorb moisture and prevent condensation on the electronics. As temperatures fluctuate day/night, moisture can condense internally – dessicant or an anti-condensation silica gel will help. If the enclosure is air-tight, also consider a **breather vent** (a Gore-Tex vent plug) which lets air pressure equalize without letting water in; this can reduce stress on seals during temperature changes.

- **Corrosion Protection:** Ocean saltwater is corrosive to metals. Use **stainless steel or brass hardware** for any exposed bolts or fittings. If you have leads or connectors that might get wet, use dielectric grease on them. Internally, any salt that enters can corrode circuit boards – hence conformal coating (acrylic or silicone conformal spray) on PCB components is a good safeguard. Pay special attention to the battery connections and any switches: a power switch exposed to salt air will likely fail; it's better to design the circuit to automatically start when the battery is connected or charged, rather than relying on an external toggle.

- **Thermal considerations:** If the buoy is in sun, the interior can heat up significantly. While most electronics operate fine up to 50°C or so, the battery life may shorten at high temp. White or bright-colored enclosures can reflect more sunlight (the Smart Buoy was painted a light color after epoxy coating [54]). Internally, avoid packing components that heat (voltage regulators, CPU) next to the

battery. Given the water around, some heat will conduct away, but if the device is sealed, it's like a mini-greenhouse. Thermal padding or placing the battery close to the hull can allow the water to help cool it.

In summary, **test your enclosure thoroughly** before deploying: perform a dunk test, leave it in a bucket overnight, check for leaks. Also test that it floats as expected (with all components inside, in both calm and agitated water). Attaching a brightly colored flag or reflective tape can help in locating the buoy from a distance – a practical detail especially near a public beach (and it alerts boaters to avoid it).

Sampling Rate and Sensor Calibration

Getting meaningful wave data requires choosing an appropriate sampling strategy and calibrating the sensors:

- **Sampling Frequency:** Ocean waves in "calm to choppy" conditions have periods on the order of a few seconds (0.2–1 Hz frequency for dominant waves). A **sampling rate of 5–10 Hz** could capture the basic wave shape. However, to give the appearance of continuous motion and to capture any higher-frequency jostling of the buoy, a higher rate is desirable. The user's intuition of <10 ms (~100 Hz) sampling is a safe choice – it ensures even rapid accelerations are recorded. In practice, many wave buoys record at 4–10 Hz to characterize waves [44], but since MEMS sensors and SD storage today can handle 100 Hz easily, we can oversample and later down-sample or filter in software. **Trade-off:** higher sampling means more data (roughly 100 MB/day logging 6 channels at 100 Hz) and slightly higher power use. If power or storage is constrained, one could sample at, say, 20 Hz – still well above Nyquist for typical waves – and reconstruct an adequate waveform. Another approach is **event-triggered sampling**: continuously sample at 100 Hz but only log (or transmit) data when motion exceeds some threshold (e.g. a wave is passing). Given the user wants a continuous motion profile, it's likely simplest to just record at a steady high rate and handle the large data volume.

- **Duty Cycling:** If battery life is an issue, consider if the buoy needs to record continuously or if periodic bursts suffice. For example, some wave buoys record 20 minutes of data every 3 hours (this is common in scientific buoys to conserve power, capturing wave stats intermittently). But if the goal is to capture the full motion for 3–4 weeks, then continuous logging is needed. In that case, adding solar or a larger battery is necessary as discussed.

- **Sensor Calibration:** MEMS gyros and accelerometers have biases (offsets) and scale factors that should be calibrated for accurate results. Before deployment:

- **Accelerometer:** With the unit stationary, read the accelerometer outputs. They should read ~[0,0,+1g] (in some orientation) for gravity. Any constant offset can be recorded. You can do a 6-point calibration (point each axis up/down to get biases and scale). Many IMUs will be close enough by default for qualitative wave data, but calibration improves accuracy especially if integrating to get velocity/displacement.
- **Gyroscope:** Gyros should read zero when still. So keep the buoy still and note the raw bias for each axis; the software can subtract these to zero the rate. Gyro drift over time is a concern – high-end systems use the magnetometer or gravity reference to correct gyro drift (sensor fusion). If not doing full fusion on the microcontroller, one can at least periodically zero the gyro if the buoy is known to have moments of no rotation (perhaps averaging over a long period to assume net rotation is zero).

- **Magnetometer:** If using it for orientation, perform a **soft-iron/hard-iron calibration** – basically rotate the device 360° in all axes in a clean environment, and use calibration software or libraries (like those that compute the offset and scale of the magnetometer readings). Since our buoy is small, we might skip magnetometer usage unless aiming for wave direction. The Smart Buoy team attempted to use accelerometer + compass to infer direction of wave travel [55], but it's complex and requires the buoy not to spin freely. It might be beyond our scope to measure wave direction accurately; most DIY efforts found it challenging [56].

- **Pressure Sensor:** If using a barometer for wave height, calibrate its reading to known atmospheric pressure at deployment time (or relative altitude). The MS5611, for instance, can resolve ~10 cm changes. You would convert pressure to altitude (using standard formulas or the sensor's internal calculator). The Smart Buoy used changes in this "altitude" over time as the wave height signal [9]. It's important to remove any slow drift (e.g. due to weather changes in air pressure) – a high-pass filter or subtracting a running mean will isolate the wave-induced pressure fluctuations.

- **Data Calibration and Filtering:** After deployment, the raw data might need filtering. Waves can be extracted by band-pass filtering the acceleration in the wave frequency band. Also, integration of acceleration to velocity or displacement is sensitive to drift – you might apply a high-pass filter to acceleration to remove the DC bias before integrating. The OpenWave project tackled this by segmenting the data at zero-crossings to avoid unbounded drift in integration [30] [57]. If computing wave height in real-time on microcontroller, a simpler method is the barometer approach or measuring tilt for wave slope. Otherwise, plan to do detailed processing on a computer after retrieving the SD card.

- **Timekeeping:** Use the RTC to timestamp data so that if you have gaps or need to sync with other observations (like a reference measurement or tide data), you can. Ensure the RTC is set to UTC or your local time and note the timezone.

In summary, sample fast enough to capture the motion (100 Hz is fine) and calibrate the sensors to minimize error. Given that our waves of interest are not extremely high-frequency, we have some freedom to reduce data later if needed.

Software and Data Management

Developing the firmware (the code on the microcontroller) and managing the data is the next crucial step:

- **Firmware Implementation:** Whether using Arduino (C/C++) or MicroPython, the code will need to initialize sensors, read data in a loop, and log or transmit it. Pseudocode for an Arduino-like approach might be:

```
setup() {
   initRTC();
   initIMU();  // e.g., start I2C, configure IMU full-scale ranges
   initSD();
   initRadio();  // if using wireless
   // maybe store start time from RTC
}
```

```
loop() {
   unsigned long t = millis();
   readIMU(&ax, &ay, &az, &gx, &gy, &gz);
   readPressure(&p);
   readBatteryVoltage(&v);
   // Format data (e.g. CSV line: timestamp, ax,ay,az,gx,gy,gz,pressure,volt)
   char line[100];
   snprintf(line, sizeof(line), "%lu,%d,%d,%d,%d,%d,%d,%.2f,%.2f\n",
            rtc.now().unixtime(), ax, ay, az, gx, gy, gz, pressure_altitude, v);
   sd_write(line);
   if (shouldTransmit(t)) {
       radio_send(packet);
   }
   // Delay or use timer to achieve ~100 Hz loop, e.g.:
   delay(10);
}
```

In practice, using interrupts or timer peripherals can yield more precise timing than `delay()`. One could set up a hardware timer to trigger IMU reads at 100 Hz. However, a simple loop with a short delay (10 ms) can be sufficient for our needs (the actual loop execution time should be measured and subtracted if precision is needed).

If using **MicroPython** on an ESP32 or Pico, the structure is similar but writing to SD card and accessing hardware will use different libraries (e.g. `machine.I2C` for sensor, Python's `open()` for file writing). MicroPython might struggle with very high frequency loops due to interpreter overhead, but 100 Hz (0.01 s loop) might still be okay if the operations are efficient (some microseconds per loop). If performance is an issue, circuitPython or MicroPython might require dropping to C for critical parts or using asynchronous programming. But for ease of development, Python is attractive – especially for formatting data or perhaps buffering to memory.

- **Data Format:** Storing data in a structured format is important. CSV text files are human-readable and easy to parse later (e.g. `time,ax,ay,az,gx,gy,gz,...`). They do incur some storage overhead compared to binary. Alternatively, logging binary (e.g. 2-byte integers for each sensor reading) is much more efficient (perhaps half the size of CSV or better), but then a custom tool is needed to decode it off-line. For 3-4 weeks of data at high rate, a binary format with a simple header might be wise to conserve space. A compromise is to log in binary during deployment and have the microcontroller convert to CSV on a portion of memory when requested or after retrieval (if the user connects via serial or Wi-Fi). But given SD cards of many gigabytes are cheap, a text format is fine as long as the card capacity is sufficient (e.g. 4 GB should hold on the order of a month of 100 Hz data, as estimated earlier).

- **Real-Time Data and Dashboard:** If implementing wireless transmission, you'll also need software on the **base station**. For example, a Raspberry Pi can run a Python script that listens to LoRa packets (via a serial or SPI interface to a LoRa module or hat) [58]. It can then log those to a file or database, and optionally provide a live dashboard. The Smart Buoy team built a web dashboard (using Vue.js for frontend) to display wave height, period, etc., updating with incoming data [59] [60]. For a DIY project, one could simply have the base station log data and perhaps use MATLAB, Python (numpy/

pandas), or Excel later to analyze it. If real-time visualization is desired, lightweight options include plotting with Python's matplotlib or using Node-RED dashboards.

- **Example Code Availability:** Many of the projects cited have published code. For instance, T3chFlicks provided their Arduino code on GitLab [61]. It includes reading the GY-86 module via I2C and computing wave parameters. Using such examples as a starting point can accelerate development. Adafruit and others also provide Arduino libraries for BNO055, MPU6050/9250, etc. – these handle the I2C/SPI communication and give you sensor values with a simple API. Just be mindful of memory impact on smaller MCUs.

- **Data Pre-processing:** If the microcontroller has headroom, you could perform some processing on the data in real-time. For example, you might compute a rolling average or filter to reduce noise, or compute wave peak-to-peak values on the fly. This can be useful for transmitting summaries (like send only the wave height and period, rather than raw data). However, for comprehensive analysis, it's usually best to store raw data and apply filters offline. One real-time calculation that is simple and useful is **battery management**: monitor the battery voltage, and if it falls below a threshold, you could reduce sampling rate or transmission frequency to conserve power, or send an alert packet indicating low battery.

- **Redundancy and Fault Handling:** Plan for potential issues – e.g., what if the SD card fails or fills up? The code could, for instance, start overwriting old data (circular buffer) or at least flash an LED to indicate a problem. If using transmission, ensure the code doesn't block endlessly if the radio hangs (implement timeouts). A watchdog timer that resets the MCU if it becomes unresponsive is a good idea for long unattended runs.

Design Pathways: Minimal vs. Enhanced Buoy Builds

Based on the above considerations, we can outline two example build scenarios:

1. Minimal Viable Offline Buoy (Budget-Focused Logger)

This design emphasizes simplicity and low cost, suitable for capturing data locally without complex extras:

- **IMU Sensor:** MPU-6050 (6-DOF) or MPU-9250 (9-DOF) module. These are cheap (≈ $5-$15) and provide raw accelerations and angular rates. For our purposes, the **MPU-6050 plus a separate barometric sensor** (like BMP280 or MS5611 ~$5) would allow wave motion + height logging on a budget. The GY-86 board used in Smart Buoy conveniently combines MPU6050 + magnetometer + barometer [3] (~$10 online).

- **Microcontroller:** Arduino Pro Mini 3.3 V @8 MHz (clone ~$5) or equivalent. It's small and can run directly off a Li-ion cell. Program it with Arduino C++ for efficient, lightweight code. Optionally use a DS3231 RTC module (~$2) for timestamps and timed sleep/wake. The Pro Mini can be put into deep sleep at a few microamps between measurements if needed (using the Narcoleptic or LowPower libraries).

- **Data Storage:** microSD card module (~$3) with a 8–16 GB microSD (~$5). Log data in binary or CSV. Format: timestamp + 6 axes + pressure. Given the Arduino's limited RAM, you might log line-by-line (ensuring to flush periodically), or use a 512-byte buffer.

- **Power:** Single 18650 Li-ion 3.7 V cell (~$5) with a step-down regulator to 3.3 V if needed (or direct if using low-dropout regulator on Pro Mini board). A small TP4056-based USB charger module ($1) can be included to recharge the battery between deployments. To last ~4 weeks, assume average current ~1–2 mA (which is attainable by mostly sleeping and waking at 10 Hz or duty cycling sensors). A 3400 mAh 18650 could then last roughly 1700–3400 hours (70–140 days) theoretically, so in practice 4 weeks is comfortable. This design might not need solar if power usage is low.

- **Enclosure:** A section of PVC pipe (~5 cm diameter) with two end caps could house the electronics and battery. One end cap can be glued shut and the other end cap sealed with an O-ring so it's openable. Alternatively, a waterproof project box (~$10) or a reused plastic bottle (with epoxy seal) can be used. Foam pool noodles or a fishing buoy float can be attached to provide buoyancy if the enclosure itself isn't buoyant enough. Make sure to mount the IMU securely inside (e.g. hot glue it to the interior). Attach an eyebolt or strong zip-ties to connect an anchor line.

- **Anchoring:** A simple weight (a small kettlebell or a sand-filled jug) tied with polypropylene rope can serve as an anchor. Tie the rope to the buoy (through the handle or an eyelet on the enclosure) and include a small slack so the buoy can bob with waves. If in swimming distance, you might not need a surfacing marker, but it's wise to add a second small float or flag so you can locate the anchor line when you want to retrieve it.

- **Cost Estimate:** Roughly summing: IMU $10, Arduino $5, sensors $5, SD + RTC $5, battery + charger $6, enclosure materials $10, misc. wiring/sealant $5. **Total ~ $40 (≈ £30)**. This minimal design doesn't have remote data access, but it's robust in that all data is on the SD card as long as the unit stays waterproof. It's ideal if you can deploy and then retrieve the buoy after a few weeks to download data.

- **Data Example:** You would get a CSV file with columns like: time, ax, ay, az (in m/s^2 or raw units), gx, gy, gz (°/s or raw), pressure (hPa) etc., sampled say 50–100 times a second. From that, you can plot the vertical acceleration or calculated altitude to see wave patterns, and use timestamps to correlate with actual events (e.g. check the period between peaks to derive wave period, use the pressure sensor to estimate wave height differences).

2. Enhanced Networked Buoy (Solar-Powered with Telemetry)

This design adds features for longer deployment and real-time data feedback. It assumes a higher budget and more complexity:

- **IMU Sensor:** Bosch BNO055 (~$35 for Adafruit breakout) or BNO080 (~$15 on breakout) for on-chip sensor fusion. This provides tilt-corrected acceleration and orientation at 100 Hz [27], simplifying computations. Alternatively, an MPU-9250 with a complementary filter in software could be used if one prefers lower cost and can handle coding. We will also include a **high-resolution barometer** (e.g. MS5611 or BMP388) for wave height measurement, and possibly a **DS18B20 waterproof temperature sensor** (~$3) if logging water temp is desired (as in Smart Buoy).

- **Microcontroller:** Espressif ESP32 (e.g. a Wemos Lolin32 Lite or TTGO LoRa32 board, ~$15). This choice gives both ample processing power and built-in wireless: some models include LoRa radios and even GPS. For example, the **TTGO T-Beam** board (~$30) has an ESP32, LoRa module, and GPS in one, plus a battery connector and charger. Using such an integrated board reduces wiring. If LoRa is not built-in, a separate LoRa module (HopeRF RFM95, ~$10) can be connected via SPI. The ESP32 will run the main code (likely in C++ using Arduino framework, or MicroPython if preferred, though C++ is recommended for efficiency). It can interface with all sensors easily and also log to SD (some ESP32 boards have an SD slot, or use an SPI SD module). We also add a **DS3231 RTC** ($2) for timekeeping and to timestamp data (the ESP32 has an internal RTC which is pretty good when synced to GPS or set at start, but an external one can be a backup).

- **Data Handling:** The ESP32 can multitask: one core can sample sensors and fill a buffer, while the other core handles LoRa transmissions or SD writing. Data strategy:

- **Logging:** Save raw or processed data to SD card continuously.
- **Transmission:** Every few seconds, send a LoRa packet containing summary stats (e.g. current wave height, period, battery voltage, GPS location). The base station onshore receives these and perhaps plots a live graph. LoRa bandwidth is limited, but sending ~50-byte packets at e.g. 2 Hz is feasible.

- If within Wi-Fi range at times, the ESP32 could also store data and when near a known Wi-Fi (like when you bring it ashore) auto-upload to a server, but that's optional.

- **Power System:** A 18650 Li-ion (or better, a 18650-size LiFePO4 cell ~1500 mAh) is used along with a **solar charging circuit**. A 5 V 2–5 W solar panel (size maybe 10×15 cm) mounted on top of the buoy can generate a few hundred mAh per day (depending on sun). The charging circuit (e.g. CN3065 or MCP73871 for LiPo, or specialized LiFe charger) manages the solar input. Include a **step-up converter** if needed to power the LoRa and other 5 V devices; however, most can run at 3.3 V so it might be simpler to keep everything on the battery voltage. For instance, the BNO055 and barometer operate at 3.3 V, the LoRa is 3.3 V, and the ESP32 is 3.3 V. So, you can avoid multiple regulators – just ensure the battery doesn't sag below ~3.0 V. The buck/boost converter in the Smart Buoy was tuned to 6 V for some reason (perhaps for charging or certain sensors)【32†】, but we might not need that complexity if using Li-ion direct. We will, however, include a **power monitor** (INA219 or similar ~$3) to log battery voltage (and solar current if desired) so we know the power status in data.

- **Enclosure & Buoy Structure:** Given more components, use a slightly larger enclosure. One approach: a strong **ABS electronics enclosure** (say 15 cm × 15 cm × 10 cm, ~$20) with gasket. Mount the solar panel on top of it (there are thin-film panels that can be epoxied on, or create a platform). The LoRa/GPS antenna should protrude – you can use an SMA bulkhead connector on the enclosure. Ensure the antenna is above water when the buoy is bobbing (might mount it on a short mast). For buoyancy, you could attach closed-cell foam around the box or put this box inside a hollow fishing buoy. Another approach is to replicate the Smart Buoy sphere: 3D-print a sphere or cylinder and waterproof it. For now, let's assume an enclosure + foam float ring for simplicity. This design will be heavier (~0.5–1 kg with all parts), so a larger buoyancy volume is needed (e.g. a 20 cm diameter foam ring).

- **Costs:** This enhanced build might total: IMU $30, ESP32/LoRa board $20, GPS $10, solar panel $15, charger and regulators $10, battery $10, enclosure/foam $20, misc (cables, sealant, antenna) $10. That's on the order of **$100–120 (≈ £80–95)**. One could trim costs by using cheaper IMU (MPU9250) and omitting GPS, etc. or by leveraging integrated boards as mentioned.

- **Capabilities:** This buoy should be able to run indefinitely in sunny conditions, and at least several days without sun. It will provide live data: for example, on your laptop or Pi base station you could see that the waves have an amplitude of, say, 0.3 m and period of 8 s currently. If something goes wrong (water intrusion, battery dying), you might notice data stops or battery voltage drops via the telemetry. Importantly, if the buoy breaks free and drifts, the GPS data could help recover it (assuming it stays within LoRa range or perhaps logs positions to SD). For safety, you might also put a cheap Bluetooth tracker or a phone number on the buoy in case someone finds it.

Both designs should incorporate **fail-safes**: for instance, if the SD card fills up, the system might overwrite oldest data or at least stop gracefully. If the battery is very low, it could cease transmissions to save the last logging capability. Testing the full system on land (or in a pool) before real deployment is highly recommended.

Bill of Materials & Budget

Below is an annotated list of components for each example build, with rough 2025 prices (USD, with GBP/EUR equivalents where appropriate):

Minimal Offline Logger Buoy – Parts List:
- **Microcontroller:** Arduino Pro Mini 328P (3.3 V, 8 MHz) – ~$5 (clone board). *(Low-power MCU to read sensors and log data.)*
- **IMU Sensor:** GY-86 10DOF module (MPU6050 accel/gyro + HMC5883L mag + MS5611 barometer) – ~$10. *(All-in-one sensor board used in Smart Buoy, provides motion and pressure for wave height* [3] *.)*
- *Alternative:* MPU-6050 module ($3) + BMP280 barometer ($2) – similar functionality without compass.
- **Storage:** MicroSD Card module (SPI interface) – $3; + 16 GB microSD card – $5. *(Stores logged data; high endurance SD recommended for frequent writes.)*
- **Real-Time Clock:** DS3231 RTC module – $2. *(Keeps time for timestamps and allows scheduling sleeps.)*
- **Battery:** 1× 18650 Li-ion 3.7 V cell (e.g. 3400 mAh) – $5; 18650 holder or tabs – $1. *(Primary power source; provides ~12 Wh of energy.)*
- **Charger:** TP4056 Li-ion USB charger board – $1. *(For recharging the 18650 via USB between deployments.)*
- **Voltage Regulator:** MCP1700-33 LDO regulator – $1. *(Optional, if Pro Mini doesn't have one or for cleaner 3.3 V supply when battery full at 4.2 V.)*
- **Enclosure:** 5 cm ID PVC tube (20 cm length) – $5; PVC end caps and O-ring – $5. *(DIY waterproof housing. One cap can be glued, the other removable with O-ring seal.)*
- *Alternative:* IP68 waterproof project box (10×10×5 cm) – ~$15 (ready-made seal).
- **Buoyancy & Mounts:** Foam float or fishing buoy (small, 15 cm) – $5; Epoxy, sealant, zip ties – $5. *(Ensures the device floats and stays upright; epoxy for sealing cable entry.)*
- **Anchor & Line:** 2 kg weight – ~$0 (repurposed item, e.g. dive weight or concrete block); 5 m of polypropylene rope – $3; small buoy for marker – $3. *(Anchoring system for stationary deployment.)*
- **Miscellaneous:** Jumper wires, PCB or protoboard for mounting components – $5; Desiccant packs – $1. *(Wiring the system and moisture control inside enclosure.)*

Estimated Cost (Minimal Buoy): **~$45 USD** (≈ £35 or €40). Using spare materials or cheaper alternatives can reduce this. The most expensive single item is the sensor module, but it's still under $15.

Enhanced Solar/Telemetry Buoy – Parts List:
- **Main Controller & Radio:** TTGO LoRa32 (ESP32 + LoRa) board – $20. *(ESP32 microcontroller with integrated LoRa radio and OLED screen; simplifies wiring.)*
- *If separate:* ESP32 dev board ($8) + RFM95 LoRa module ($10) + SMA antenna ($5).
- **IMU Sensor:** Adafruit BNO055 9DOF breakout – $35. *(Self-fusing IMU for ease of getting orientation and linear acceleration [27].)*
- *Alternative:* CJMCU-117 (BNO080 breakout) – ~$15 (cheaper, newer Bosch IMU).
- **Pressure/Temp Sensor:** MS5611 barometer module – $5. *(High-resolution pressure for altitude changes i.e. wave height; also measures air temp.)*
- **GPS Module:** u-blox NEO-6M or -8M GPS with antenna – $10. *(Provides coordinates and timestamp; useful for drift tracking and time sync.)*
- **SD Storage:** MicroSD slot (if not on the dev board) – $3; microSD card 16 GB – $5.
- **Power Management:**
- 2× 18650 Li-ion 3.7 V cells in parallel – $10 (or 2× 3.2 V LiFePO4 1500 mAh – $12). *(Provides ~6000 mAh total for high endurance.)*
- Solar panel, 5 V 3 W (roughly 15×15 cm) – $15. *(Generates ~500 mA in bright sun; mount on buoy.)*
- Solar charge controller (Li-ion MPPT, e.g. CN3065 module or Adafruit Solar Charger) – $10. *(Charges battery from panel efficiently, prevents overcharge.)*
- DC-DC buck converter (3.3 V step-down if needed for stable voltage) – $5. *(Ensures clean 3.3 V to sensitive electronics when battery is above 3.3 V.)*
- Power switch or latch (optional, for manual on/off) – $3; INA219 or similar voltage/current sensor – $3 (monitors battery and solar current).
- **Enclosure:** Polycarbonate or ABS waterproof case (~20 cm × 20 cm × 10 cm) – $20. *(Houses all electronics; transparent lid if solar panel needs sun, or mount panel externally.)*
- **Buoy Structure:** EVA foam or polyurethane foam ring (20–30 cm diam.) – $10. *(For flotation around the case; or use a ready-made buoy housing.)* Epoxy, marine sealant – $5.
- **Hardware:** Stainless steel U-bolt or eye bolt for anchor attachment – $3; antenna mount (bulkhead) – $2.
- **Anchor & Retrieval:** Similar to minimal – rope, weight, marker buoy – $5. Perhaps add a small **strobe light or reflector** on the buoy for visibility – $5.

Estimated Cost (Enhanced Buoy): **~$130 USD** (≈ £100 or €120). Costs can be optimized by using what's on hand (e.g. if you have an old solar garden light for panel, etc.). This build invests in capabilities (solar, comms, GPS) that drive the price up.

Both cost estimates exclude labor and any 3D printing or tools. The enhanced version is approaching a semi-professional instrument, albeit still far cheaper than commercial wave buoys (which run in the thousands of dollars [24]).

Deployment, Recovery, and Maintenance

When it comes time to put the buoy in the water, and later retrieve it, consider these practical tips:

- **Deployment:** Choose a calm day for deployment to safely swim or boat the buoy out. Since this buoy will be near Margate's shore (swimming distance), you might wade out at a lower tide or paddle a

kayak to the deployment spot. If using an anchor, gently lower the anchor weight to the seabed to avoid tangling the line, then release the buoy. Ensure the rope length is slightly greater than the water depth (to account for tides and wave motion). If in tidal waters, include extra slack or a small buoy on the line to keep it from yanking the main buoy down at high tide. The buoy should be free to bob on waves but not drift away.

- **Marking and Visibility:** In coastal waters, it's important to mark the buoy so it's visible to boaters and jet-skiers. A bright color (yellow or orange) is typically used for scientific buoys. You could add a flag or reflective tape. Also, write contact info on it in case it's found adrift. In some regions, deploying any object in navigable waters might require permission if it's large or long-term; a small buoy near a beach for a short project is usually fine, but checking local regulations is wise.

- **During Operation:** Monitor from shore if possible. With the enhanced buoy, you can receive data – a sudden change or flat-lining might indicate an issue (e.g. water ingress causing sensor failure, or the buoy capsized). With the basic buoy, you have no data until recovery, so perhaps consider a quick check after the first 24 hours (retrieve and see if data is logging correctly, then redeploy) to ensure all is well, before leaving it out for weeks.

- **Recovery:** Plan how to retrieve the anchor. Pulling by rope from shore works only if you can get an angle; otherwise you may need to swim/dive to free it. It helps to include a small surface float directly above the anchor (so you can find that spot and pull vertical). Rinse the buoy with fresh water after recovery to get rid of salt. Immediately remove the SD card and back up the data. Check the interior for any signs of leakage or corrosion.

- **Maintenance for Redeployment:** After each deployment, replace corroded parts, recharge or swap batteries, and check seals. Desiccant packs should be re-dried or replaced. If using solar, clean the solar panel surface. For long-term multi-deployment use, consider applying a thin coat of anti-fouling paint on the hull to reduce algae growth if the buoy stays out for more than a few weeks (marine growth can add weight and change buoyancy over time, and make a mess of your enclosure).

- **Data Analysis:** Once you have the data, you'll likely want to analyze wave characteristics. You can calculate the wave height time series from the barometric data (subtract the mean pressure, convert to relative height). You can also integrate the vertical acceleration (with corrections) to estimate displacement. For wave period, an algorithm is to find the zero-crossings or use an FFT to identify the dominant frequency. The Smart Buoy team, for example, used a midpoint crossing method to get period [10]. If you have directional data (perhaps from magnetometer or two buoys), that gets complex, but for one buoy you might at least categorize wave headings qualitatively.

Failure Points and Mitigations

Finally, it's important to acknowledge where things might go wrong and how to mitigate them:

- **Water Leakage:** The foremost failure mode for any ocean device. Even a few drops of saltwater can short electronics or ruin a battery. Mitigations: double-seal critical points (e.g., O-ring plus silicone grease, then a secondary outer seal tape). Use a conformal coating on PCBs so that if a tiny leak

occurs, the circuits have some protection. Monitor for condensation – if you see fog inside the enclosure, it's a warning sign to improve the seal or add more dessicant.

- **Corrosion:** Salt air corroding connectors, wires, and boards is a slower issue. Mitigations include using corrosion-resistant materials (stainless steel, gold-plated contacts) and applying corrosion inhibitor sprays internally. If the buoy is used over many months, consider periodically opening it up to inspect and clean any rust.

- **Power Depletion:** The buoy could run out of battery earlier than expected (perhaps due to higher power draw or lack of sun). If it dies, data logging stops (and in a worst case, an SD write might corrupt the last file). To prevent data loss, code the microcontroller to close the file regularly (say every hour or even every few minutes) so that if power is lost the file isn't left open. Also include a voltage cutoff – e.g., if battery falls below safe level (like 3.0 V for Li-ion), stop logging to preserve the SD card integrity and avoid over-discharging the cell. With solar in the enhanced version, ensure the charge controller has over-discharge protection or use a protection circuit on the Li-ion battery.

- **SD Card Issues:** SD cards can fail or get corrupted, especially if power is lost during a write. Use high-quality cards and implement proper file handling. Mitigation: the MCU can maintain an index and write data in append mode, so even if one write is bad, previous data is intact. Using a simpler format (like writing fixed-size binary records) can allow data recovery even if a FAT filesystem is corrupted. The Smart Buoy team abandoned SD partly due to complexity [36]; if one is concerned, an alternative is an FRAM chip to store data, but as discussed, capacity is limited. So with SD, just handle with care in software.

- **Loss of Buoy (Drifting Away):** If the anchor fails (rope breaks or comes loose), the buoy may drift off. This is a real risk; mitigations: use a strong marine-grade rope, maybe double up critical knots or use metal crimps. Adding GPS and a way to transmit position (even via a periodic SMS if using cellular, or via LoRa to any listening station) greatly aids recovery. A low-tech solution is to write "REWARD if found, call XYZ" on the buoy. Since our scenario is near shore, having a second small float attached by a few meters of line can act as a backup buoyancy if the main one somehow floods and sinks (the backup float could keep it at surface).

- **Sensor Failure or Drift:** Over weeks, sensor drift (especially gyro drift) could accumulate in data. If using a fused sensor like BNO055, it should auto-calibrate with motion, but it might still wander. In post-processing, one can often correct drift by high-pass filtering out any very low-frequency bias. If an accelerometer axis fails or saturates, you might lose data on that channel. To catch such issues early, it's good to have some redundancy – e.g. the barometer gives an independent measure of vertical motion that can be compared to accelerometer-integrated displacement to validate it. Or two different sensors (if budget allowed two IMUs) could cross-check (unlikely in DIY but conceptually).

- **Computational Errors:** An overlooked risk is a bug in the code causing a crash or memory overflow after some days (e.g. an unsustainable memory leak in Python, or an int32 timer overflow). Testing the code for extended periods (run it logging to a file for a few days on the bench) can reveal these. On Arduino, `millis()` overflows after ~50 days – which is just beyond our 4-week window, but if the buoy ran longer that could cause timing issues if not handled. Use a watchdog timer to auto-reset if the software hangs. On ESP32, be mindful of task watchdog if using the RTOS features.

- **Environmental Factors:** The sea can be unpredictable. Large waves could submerge the buoy briefly – electronics should handle that (pressure sensor might go out of range if suddenly underwater, but it should recover). If the buoy gets fouled by seaweed or hit by a bird or boat, it could affect orientation or damage parts. Hardening the structure (using a cage or bumper around delicate parts like the solar panel) can help. UV sunlight degrades plastics – ensure any 3D printed parts are UV-stabilized (coatings or ASA/PETG materials). Temperature in winter could affect battery performance (Li-ions lose capacity in cold); if deploying in cold season, consider insulation or using LiFePO4 which handles cold better.

By anticipating these issues, a DIY buoy builder can improve the odds of a successful deployment and quality dataset. It's all about balancing cost and complexity with reliability: for critical measurements, you might add more safety (more sensors, redundant logging, stronger build) at the expense of budget. But even a modest build, if well thought out, can provide invaluable insight into wave dynamics.

Suggested Improvements and Variations

Once the basic buoy is working, there are many possible enhancements or experiments to explore: - **Multiple Buoys / Distributed Measurements:** Building two or more buoys would allow studying wave propagation (e.g. measure phase lag between buoys a known distance apart to estimate wave direction and speed). This could be a larger project but yields more data. Each buoy could transmit to the same base station on different IDs. - **Advanced Sensing:** Incorporate a water level sensor such as an **ultrasonic ranger** or LiDAR facing downward to directly measure wave surface (this works only if the sensor stays a fixed height above water, which a buoy does not, so it's more for fixed platforms). Or use a **water pressure sensor** on a tether below the buoy to measure wave pressure fluctuations under water (common in oceanography to measure wave height from the seafloor). These could augment the IMU data. - **Energy Harvesting:** If solar is not sufficient or feasible (say in high latitudes winter), one could experiment with wave energy harvesters (point absorbers) or just add a wind turbine (if often windy). This gets complex, but even a small vertical-axis turbine could trickle-charge the battery when waves are accompanied by wind. - **Real-time Processing:** Implementing some processing on the microcontroller, like computing FFT spectra of the wave motion in real-time, could be a fun challenge. The ESP32 likely has enough power to do an FFT on 1024 samples and transmit a wave spectrum instead of raw data, for example. - **Alternative Communications:** Trying out a 4G LTE-M or NB-IoT module could let the buoy send data directly to the internet (useful if you want it to tweet wave heights or send to a cloud database). This would remove the need for a local receiver but requires ensuring cell coverage and managing power for the modem. - **Machine Learning on board:** For the adventurous, one could use a tiny ML model on the device to classify sea state (e.g. calm, moderate, rough) from the motion data. This could be transmitted as an intelligible metric rather than raw numbers. - **Longer Deployments:** With some modifications (bigger battery, robust build), you might attempt multi-month or year-round deployments. Maintenance like periodically cleaning solar panels and checking seals would be needed. Using LiFePO4 batteries is advisable for many cycles and safety. Ensuring everything is double-encased (maybe the electronics in a watertight inner box inside an outer buoy) can give more confidence for long term.

In essence, this DIY buoy project is highly modular – you can start simple (just record motion to an SD card) and gradually add complexity (telemetry, solar, more sensors) as you gain experience and confidence. It exemplifies a blend of disciplines: electronics, programming, and marine engineering. By learning from prior projects and following best practices in design and calibration, a hobbyist can create a functional

wave-sensing buoy that yields valuable environmental data and survives the challenges of the open water. Happy buoy building!

References: The insights and examples above reference several sources, including open-source buoy projects [13] [62], academic research on low-cost wave buoys [18] [1], and DIY tutorials [4] [63]. These demonstrate the feasibility of using inexpensive IMUs for wave measurement and offer practical guidance on construction, power management, and data analysis in real-world conditions. All cited works are credited inline to acknowledge their contributions to this comprehensive overview.

---

[1] [18] [19] [20] [21] [22] [23] [33] [34] [43] [44] [50] [51] [52] researchgate.net
https://www.researchgate.net/profile/Yury-Yurovsky/publication/323354914_Compact_low-cost_Arduino-based_buoy_for_sea_surface_wave_measurements/links/5a93ea4545851535bcd983c2/Compact-low-cost-Arduino-based-buoy-for-sea-surface-wave-measurements.pdf

[2] [24] [25] [26] [27] [28] [29] [30] [32] [35] [57] OpenWave : Development of an Open-Source and Low-cost wave sensor – Wattnotions
https://wattnotions.wordpress.com/2019/02/02/development-of-an-open-source-and-low-cost-wave-sensor/

[3] [4] [9] [10] [12] [45] [55] [63] Smart Buoy — Making Wave and Temperature Measurements | by T3ch Flicks | Nerd For Tech | Medium
https://medium.com/nerd-for-tech/smart-buoy-making-wave-and-temperature-measurements-%EF%B8%8F-cdda14c52196

[5] [6] [7] [8] [11] [31] [36] [37] [38] [39] [40] [56] [62] Smart-Buoy | Hackaday.io
https://hackaday.io/project/166572-smart-buoy

[13] [14] [15] [16] [17] [58] Arduino | Open Source Ocean Data Buoy Project
https://opensourceoceanweatherbuoy.wordpress.com/tag/arduino/

[41] [42] [46] [47] [48] [49] [53] [54] [59] [60] [61] Smart Buoy [Summary] : 8 Steps (with Pictures) - Instructables
https://www.instructables.com/Smart-Buoy/