ABSTRACT

Autonomous Underwater Vehicles (AUVs) require extremely accurate and precise velocity estimation for navigation, yet commercially available Doppler Velocity Logs commonly known as DVLs are prohibitively expensive, power-intensive as well as bulky. This research addresses the crucial gap in affordable underwater velocity measurement by implementation and development of a machine learning enhanced Differential Pressure Speedometer Sensor (DPSS) system on custom embedded hardware. The project addresses the fundamental problem of processing highly nonlinear, noisy pressure differential signals via a Long Short-Term Memory neural network on an STM32H563ZIT6 microcontroller. A custom validation setup consisting of a 6-meter linear rail system with high-resolution rotary encoder supplied the ground truth data to train the model. The system developed takes in multi-sensor measurements (two pressure differentials, IMU orientation, temperature) at 500 Hz and delivers velocity estimates at 100 Hz with sub-centimeter-per-second precision. Output shows that LSTM-based filtering performs much better compared to conventional signal processing techniques, achieving Mean Absolute Error down to 2.2 mm/s with real-time capability. Integration of the Extended Kalman Filter with IMU data further enhances velocity estimates. This research provides a low-cost practical framework for navigation in AUVs, bringing underwater robotics to research labs, schools, and developing organizations. The full hardware design, firmware code, and trained neural network models offer an implementable solution for real-world underwater robots.

Keywords: Autonomous Underwater Vehicles, Differential Pressure Speedometer, Embedded AI, LSTM Neural Networks, Real-time Signal Processing, STM32 Microcontroller, Sensor Fusion

CONTENTS

LIST OF FIGURES

LIST OF ACRONYMS

AUV	Autonomous Underwater Vehicle
DPSS	Differential Pressure Speedometer Sensor
MCU	Microcontroller unit
DVL	Doppler velocity log
GPS	Global Positioning System
ROV	Remotely Operated Vehicles
PCB	Printed Circuit Board
EMI	Electromagnetic Induction
INS	Inertial Navigation Systems
MEMS	Micro-Electro-Mechanical Systems
IMU	Inertial Measurement Unit
USBL	Ultra-Short Baseline
LBL	Long Baseline
CFD	Computational Fluid Dynamics
LSTM	Long Short-Term Memory
ADC	Analog to Digital Converter

Chapter 1

Introduction

1.1 INTRODUCTION

Oceans are still one of the least examined places on our planet despite occupying about 71% of its surface area. Studying the marine ecosystems, monitoring and investigating underwater facilities, carrying out oceanographic studies, and executing defense activities all rely on underwater vehicles, particularly those that are operable un-manned. These automated vehicles explore into ocean depths and circumstances in which human existence is inconvenient or not possible, acting as our eyes, ears, and hands under the water.

The operation of an Autonomous Underwater Vehicle (AUVs) hinges solely on its capacity to navigate efficiently through the underwater environment. In contrast to GPS-dependent positioning on land or in the air, underwater robots must contend with a special challenge: electromagnetic waves, such as GPS signals, strongly attenuate in seawater and penetrate merely a few centimeters into the water. This leaves AUVs with no choice but to rely on other forms of navigation, mostly dead-reckoning based on integrated velocity measurement. Velocity estimation accuracy thus directly influences success in the mission—mapping underwater geological features, surveying offshore oil rigs, tracking marine life, or carrying out search and rescue.

Velocity sensors currently used in industry, mainly Doppler Velocity Logs (DVLs), offer high measurement precision at high expense. A standard DVL unit costs anywhere from $15,000 to $50,000, with a power consumption of 10-30 watts, a few kilograms of weight, and the need for sophisticated mounting and integration. These forces price out a large number of prospective users: schools teaching underwater robots, hobbyists working on ocean technology, small research labs with tight budgets, and start-up companies developing innovative underwater systems. The technological capability is there, but the economics keep it from spreading.

This work answers a simple question: Can we build a reliable, real-time underwater velocity sensor at a fraction of the cost and complexity of existing commercial products? The solution is in merging traditional fluid dynamics concepts with contemporary machine learning methods, namely applying neural networks that have been found suitable for time-series data to embedded microcontroller-based hardware for the processing of signals from Differential Pressure Speedometer Sensor (DPSS) to deliver sub-centimeter-per-second velocity estimates at a cost of under $300.

The technology introduced here is more than an incremental innovation; it is a paradigm change in underwater velocity sensing. Making high-accuracy navigation affordable and accessible, this work allows more than a niche community to explore underwater robotics, driving innovation and widening the applicability space for autonomous marine systems. From class projects to start-up firms, from research vessels to classroom demonstrations, the technology introduced here unlocks doors previously shut by cost considerations.

1.2 OVERVIEW OF AUNTONOMOUS UNDERWATER VEHICLES

Autonomous underwater vehicles are a heterogeneous group of robotic platforms for underwater operation without persistent human involvement. The discipline includes a range of vehicle types, each tailored to specific mission profiles and operating demands.

Scientific Research AUVs emphasize acquiring oceanographic data, frequently burdening sensor suites to measure temperature, salinity, dissolved oxygen, chlorophyll density, and other environmental variables. They usually are operated in depths from near-surface waters down to several thousand meters, and the duration is typically in hours to weeks. Institutions such as Woods Hole Oceanographic Institution, Monterey Bay Aquarium Research Institute, and National Oceanography Centre utilize research AUVs to investigate ocean circulation, marine biology, and seafloor geology. Mission profiles tend to call for station-keeping ability to keep a steady position while gathering measurements or tracing intricate survey patterns over large expanses with low position uncertainty.

Industrial Inspection AUVs are used by the offshore energy industry, inspecting pipelines, subsea facilities, and platform foundations. The oil and gas market alone uses hundreds of AUVs every year for standard inspections that would otherwise use costly manned submersibles or surface vessels with remotely operated vehicles (ROVs). These uses call for high-resolution imaging performance, accurate positioning in relation to infrastructure, and fault-tolerant operation in difficult environments such as rough currents, limited visibility, and electromagnetic interference from adjacent electrical systems. AUV inspection of pipelines, for example, involves keeping to a straight path with enough precision to find irregularities, corrosion, or structural wear with consistent distance and orientation to the pipe.

Military and Defense AUVs are used to conduct mine countermeasures, intelligence gathering, surveillance, and harbor protection. Defense uses generally involve stealthy operation, so DVL acoustic transmissions are likely to be undesirable because they are detectable. These ships usually work in hostile or disputed areas where accurate navigation is a matter of mission success but acoustic quietness. Harbor and coastal missions take place in shallow, cluttered waters where conventional acoustic positioning can fail from reflections and interference. The

defense industry has therefore heavily invested in alternative navigation technologies, such as DPSS research.

Educational and Hobby AUVs are an increasing segment as universities include underwater robotics in engineering courses and hobbyists venture into ocean technology. Student competitions such as AUVSI Foundation's RoboSub and Singapore AUV Challenge foster innovation in low-cost underwater systems. These users have very tight budgetary constraints—average student teams work with yearly budgets less than $20,000 for full vehicle development including electronics, sensors, propulsion, and mechanisms. Commercial DVL systems therefore take up most or all the available budget, with compromises in other abilities or omission of velocity sensing altogether. This knowledge barrier discourages many prospective engineers from having hands-on training with underwater navigation technology.

1.3 UNDERWATER NAVIGATION REQUIREMENTS

Positioning Accuracy: This varies depending on the mission, but position uncertainties below 1-5 meters after minutes to hours-long missions are common. Scientific survey missions of bathymetric or environmental data must know where in space each measurement was taken. Pipeline inspection requires maintaining position within the pipe's field of view; sub-centimeter accuracy is required for applications like underwater docking or manipulation tasks. A velocity sensor with an error of 1 cm/s would then cause 6 meters of position uncertainty after 10 minutes (600 seconds). Requirements on positioning accuracy thus call for velocity measurement errors well beneath 1 cm/s for mission time spans.

Update Rate Requirements: The control system requires timely state information about the vehicle. Control loops typically execute at 10-100 Hz in the more advanced AUV autopilots, refreshing thruster commands and control surface positions based on current vehicle state. The navigation filters, usually Extended Kalman Filters, fuse velocity measurements with the inertial sensors. Velocity inputs are thus needed at comparable rates to keep the filter stable and tracking well. Lower update rates drastically reduce the control system's disturbance rejection capability and the ability to follow the desired trajectories. For instance, a velocity sensor producing updates at only 1-5 Hz forces the navigation filter to rely more on the inertial measurements between the velocity observations, thereby increasing drift between corrections. Therefore, the industry practice aims at achieving update rates from velocity sensors in the range 10-50 Hz, with higher update rates for vehicles in aggressive motion and when strong currents are present.

Integration with Guidance and Control systems requires velocity sensors to provide measurements in appropriate reference frames with well-characterized noise properties. AUV control systems typically work in body-fixed reference frames (Forward-Starboard-Down or

FSD) or earth-fixed frames (North-East-Down or NED). Velocity sensors must report measurements consistent with these conventions, properly accounting for vehicle orientation. Additionally, navigation filters need realistic uncertainty estimates (covariance matrices) for optimal sensor fusion. A velocity measurement reported with overly optimistic uncertainty will be weighted heavily in the filter, potentially causing instability if the actual measurement error is larger. Conversely, reporting excessive uncertainty reduces the sensor's contribution to state estimation, defeating the purpose of including it. Proper characterization of measurement errors and their dependencies on operating conditions is thus essential. Dead-reckoning vs. absolute positioning represents a fundamental trade-off in underwater navigation. Dead-reckoning accumulates error over time but is capable of providing continuous state estimates without the use of external infrastructure. Absolute positioning methods bound the growth of error but, in turn, impose operational constraints. The most capable AUV navigation systems combine both approaches: dead-reckoning between absolute position fixes, using velocity-based navigation when external references are unavailable or undesirable. DPSS-based velocity sensing fits naturally into this architecture, providing continuous velocity inputs that a navigation filter integrates into position estimates, with position drift bounded by occasional absolute measurements when available.

1.4 CURRENT VELOCITY SENSING TECHNIQUES

The underwater robotics community has developed a set of approaches to velocity measurement, each with different advantages and limitations that dictate their suitability to a variety of missions and platforms.

Doppler Velocity Logs (DVLs) represent the main commercial solution for underwater velocity measurement. Acoustic instruments send out several sound pulses that are usually transmitted along four beams in a Janus configuration and measure the Doppler frequency shift of acoustic returns to calculate velocity. Two main modes of operation exist: bottom-tracking mode, which measures velocity with respect to the seafloor by detecting returns from seabed reflections, and water-tracking mode, which measures velocity with respect to the water mass by analyzing backscatter from suspended particles and density variations within the water column.

However, DVLs suffer from significant limitations that motivate alternative approaches. Cost represents the primary barrier: entry-level DVLs cost $15,000-$20,000, while research-grade instruments exceed $40,000. This pricing immediately excludes many potential users. Size and weight constraints matter for small vehicles—DVLs typically measure 15-30 cm in diameter, weigh 2-5 kg, and require substantial mounting structures. A 10 kg vehicle carrying a 3 kg DVL faces significant payload penalties. Power consumption limits battery-powered mission duration. A small AUV with 500 Wh battery capacity running a 20-watt DVL expends 25 hours worth of power, substantially reducing range and endurance. Operating constraints further limit applicability: bottom-track requires sufficient altitude above seabed (minimum

typically 0.5-1.5 meters), making near-bottom operations challenging. Water-track depends on adequate particle concentration, failing in exceptionally clear waters. Mid-water column operation, neither near bottom nor near surface, may not provide reliable returns in either mode. Acoustic interference between multiple vehicles operating in proximity can cause measurement errors or complete failure, limiting multi-vehicle operations.

Acoustic Positioning Systems provide position information rather than continuous velocity but merit mention due to their common integration with DVLs in underwater navigation systems. Ultra-Short Baseline (USBL) systems mount a transducer array on a surface vessel, measuring range and bearing to acoustic transponders on underwater vehicles. The positioning accuracy is usually achieved within 0.5 to 5% of slant range, which means a vehicle at 100-meter depth may have from 0.5 to 5 meters of position uncertainty. In Long Baseline (LBL) systems, fixed transponders are deployed on the seafloor, triangulating vehicle position from ranges to multiple beacons. LBL normally offers better accuracy than USBL, often sub-meter, but requires deploying infrastructure and limits the operational area to the baseline's geometric coverage.

Both USBL and LBL provide intermittent position fixes rather than continuous velocity, requiring integration with other sensors-DVL or inertial-for complete navigation solutions. Second, acoustic positioning has some more fundamental physical limitations. The acoustic propagation velocity is temperature-, salinity-, and pressure-dependent, so a sound velocity correction is needed to ensure accuracy. In shallow water, or close to structures, multipath reflections generate measurement errors. Update rates are rarely higher than 1-2 Hz due to time-of-flight delays and signal processing requirements.

Inertial Navigation Systems estimate velocity and position by integrating accelerometer measurements over time. High-grade tactical IMUs are capable of providing position estimates accurate to meters over timescales of minutes, but such units cost $10,000 - $100,000. MEMS-grade IMUs affordable for research and education ($20-$500) accumulate errors so rapidly - meters in seconds - that standalone inertial navigation proves impractical for all but the shortest missions. INS consequently serves as a complementary sensor, providing high-rate attitude estimates and velocity updates between measurements from other sources. Industry standard practice is integration of INS-DVL via Extended Kalman Filtering, which fuses the bounded error of DVL with the high bandwidth/short-term accuracy of INS. GPS Surface Positioning obviously provides absolute position when the vehicle surfaces but offers no underwater capability. Some missions incorporate periodic surfacing for GPS fixes, bounding dead-reckoning error growth, but surfacing interrupts the underwater mission, may be operationally undesirable (detection concerns in defense applications), or physically impossible when operating under ice, in enclosed spaces, or when mission profiles require continuous deep operation. This landscape of existing technologies reveals a clear gap: no current low-cost solution provides continuous velocity measurements with the accuracy needed for practical underwater navigation. The performance of DVLs is excellent, but at prohibitive cost. Acoustic positioning requires infrastructure and lacks velocity output. INS alone drifts too rapidly. The

opportunity for DPSS technology emerges from this gap-providing velocity sensing at a fraction of DVL cost while maintaining sufficient accuracy for practical navigation applications.

1.5 DIFFERENTIAL PRESSURE SPEEDOMETER SENSORS

DPSS technology exploits fundamental relationships between fluid velocity and pressure that are described by Bernoulli's principle. As formulated in 1738 by Daniel Bernoulli, the principle states that along a streamline in incompressible, inviscid flow, the total mechanical energy remains constant:

$$p + \frac{1}{2}\rho v^2 = \text{constant}$$

where (P) is static pressure, (rho) is fluid density, (v) is flow velocity, (g) is gravitational acceleration, and (h) is elevation. For horizontal flow at constant elevation, this simplifies to:

$$P + \frac{1}{2}\rho v^2 = P_0$$

where (P_0) is stagnation pressure—the pressure at a point where flow velocity reaches zero.

A body moving through fluid at velocity (v∞) experiences stagnation pressure at its forward-facing point (where the fluid velocity relative to the body is zero) and reduced pressure at locations where the flow accelerates around the body. This pressure difference is directly proportional to the velocity. For an idealized pitot-static tube with perfectly positioned stagnation and static ports:

$$v_\infty = \sqrt{\frac{2(P_0 - P_s)}{\rho}}$$

Where P_0 is stagnation pressure at the front port and P_s is static pressure at the side port.

Real implementations introduce geometric factors and deviations from ideal flow conditions. The actual velocity relationship becomes:

$$v_\infty = k\sqrt{\frac{2\Delta P}{\rho}}$$

where k is a calibration constant that accounts for port geometry, boundary layer effects, and flow regime. Typical values for k in common DPSS applications range between 0.6 and 0.9 and are determined empirically through calibration.

Geometry is another factor. Ports with sharp edges tend to cause separation at the opening, leading to local pressure perturbations. A chamfered or recessed port minimizes such problems. The diameter of the port influences response time: larger ports yield stronger signals but respond more slowly to velocity changes because of pneumatic time constants in connecting tubing and sensor cavities.

Temperature Effects complicate the velocity-pressure relationship through multiple mechanisms. Fluid density, ρ, changes with temperature, and this directly affects the velocity calculation. For seawater, the density decreases approximately 0.2 kg/m³ per °C, creating a 0.1% error in velocity per degree if uncompensated. Pressure transducers themselves exhibit temperature-dependent sensitivity and offset. The most common type of sensor for DPSS applications, piezoresistive, may show up to 0.1-1% variation in sensitivity and several Pascal changes in offset per degree Celsius. Careful thermal characterization and online compensation prove necessary for accurate measurements across the temperature range encountered in underwater operations, often within a range of 5-30°C depending on depth and location.

1.6 ADVANTAGES AND CHALLENGES OF DPSS

DPSS technology offers very compelling benefits compared to both acoustic and inertial alternatives, hence its interest in the underwater robotics community, yet faces significant technical obstacles that have historically prevented widespread adoption. The benefits come in economic, operational, and performance dimensions, while the problems are all about signal quality and processing complexity.

From a cost perspective, a complete DPSS implementation requires two differential pressure transducers ($20-50 each), analog signal conditioning electronics ($30-50), a microcontroller ($15-30), and mechanical components (<$50), with total hardware cost typically falling below $200, potentially under $100 in volume production. This represents 1/100th to 1/200th the cost of entry-level DVL systems, transforming velocity sensing from a budget-dominating expense to an affordable component virtually any project can incorporate. Power consumption similarly favors DPSS: analog pressure transducers draw 1-5 milliamps at 5 volts (5-25 milliwatts), with digital conditioning and microcontroller processing adding 100-500

milliwatts, for total system power typically ranging 100-600 milliwatts—1/20th to 1/50th that of DVL systems. For battery-powered AUVs, this difference directly translates to mission endurance: a vehicle with 500 Wh battery capacity gains approximately 24 hours of operation by replacing a 20-watt DVL with a 0.5-watt DPSS system.

Physical integration advantages are equally significant. Pressure transducers measure 10-25mm on a side, weigh 5-20 grams, and require minimal mounting structure, with total sensor assembly mass typically falling below 100 grams including electronics, cabling, and mounting hardware, requiring only a few cubic centimeters. Small AUVs that cannot accommodate DVL packages can readily integrate DPSS sensing. Operationally, DPSS sensors contain no moving parts, require no periodic maintenance, and operate passively without transmitting acoustic signals, with no beam alignment, minimum altitude, or bottom-lock requirements. Mid-water column operation poses no special challenge, and unlike acoustic systems that may interfere with each other, multiple DPSS-equipped vehicles can operate in close proximity without interaction. For military applications, DPSS provides velocity sensing without acoustic transmission, avoiding detection risks, while in civilian applications, acoustic silence is valuable for marine biology research where active acoustic systems may alter behavior of study subjects.

Despite these advantages, DPSS has significant technical challenges. Real fluid flow deviates substantially from ideal Bernoulli conditions: boundary layers create velocity gradients from zero at the hull surface to free-stream velocity at the edge of the boundary layer. Turbulent flow (Reynolds numbers of 10^5 to 10^6 for typical AUVs) creates chaotic pressure fluctuations superimposed on mean pressure distributions; flow separation and vortex shedding drastically alter pressure distributions. The resulting relationship between velocity and pressure is nonlinear and dependent on many factors other than speed: vehicle pitch and yaw angles change the pattern of flow around sensor ports; angular rates induce additional pressure gradients; linear accelerations couple into the pressure field through inertial forces; temperature affects fluid properties and sensor characteristics. This results in a complex multi-dimensional mapping from the measurement space-two pressure values, vehicle orientation, temperature, angular rates, and accelerations-to velocity estimate, one that is difficult to characterize analytically.

Noise represents the most of the challenges for DPSS implementation. Analog pressure transducers in underwater vehicle environments suffer from electromagnetic interference due to switching electronics that couple into sensor signal paths; with transducer outputs at millivolt levels, even small interference is significant. Thruster vibrations couple into sensors through mechanical mounting of both pressure transducer and vehicle, pressure port tubing resonances, and fluid acoustic resonances; vibration spectra contain fundamental frequencies at motor rotation rates of 10s to 100s of Hz, with harmonics extending to several kHz. Temperature changes alter sensor offset and sensitivity; diving through thermoclines may change temperature by 10°C over minutes, and internal temperature can change as electronics dissipate heat. For 12-bit ADCs spanning 3.3V, each count represents 0.8 mV, so that pressure transducers with 10

mV full-scale output see quantization steps of approximately 8% of full scale. Natural turbulence creates genuine pressure fluctuations-random variations associated with turbulent eddies convecting past pressure ports-that cannot be eliminated through shielding or isolation; these typically produce pressure fluctuations with standard deviations of 1-5% of dynamic pressure. These combined noise sources often produce signal-to-noise ratios of 10-20 dB, such that pressure signals are buried in noise roughly one-tenth to one-third their magnitude. Variable operating conditions further complicate DPSS implementation. Underwater vehicles operate over wide ranges of conditions, with the velocity ranging from near-zero during station-keeping to 1-2 m/s during transit. Low velocities are especially challenging: dynamic pressures scale as v^2, so reducing velocity from 1 m/s to 0.1 m/s reduces the pressure differential by a factor of 100, pushing signals toward the noise floor. Temperature variations of 10-20°C commonly occur during deep dives, salinity changes alter fluid density by 2-3%, and vehicle attitude can vary over ±180° ranges, although typical operations remain within ±30° of level trim. A fixed-parameter processing approach cannot handle this variability optimally: filter parameters that provide adequate noise suppression at high velocity may prove too aggressive at low velocity, oversmoothing signals and introducing excessive latency; temperature compensation requires knowledge of the current temperature and appropriate corrections, sensor placement and thermal lag adding additional complexity.

1.7 OBJECTIVES

Specific objectives of this research are to:

Primary Objective: Develop and validate a complete DPSS-based underwater velocity estimation system integrating custom electronics, LSTM neural networks, and embedded microcontroller deployment that provides ≤5 mm/s velocity estimation accuracy at an output rate of 100 Hz with total hardware cost below $300.

Objective 1-Hardware Development: Design and fabricate custom printed circuit board electronics, enabling simultaneous multi-channel pressure sensor acquisition at 500 Hz with noise floor below 1 mV RMS, including STM32H563ZIT6 microcontroller integration, analog signal conditioning, and sensor interfaces using SPI, I2C, and ADC. Validate hardware performance through electrical characterization and noise spectrum analysis.

Objective 2 - Validation Apparatus Construction: Construct a 6-meter linear rail validation system that provides ground truth velocity measurements through a high-resolution rotary encoder (≥2000 counts/revolution) coupled to linear motion. The apparatus shall allow for repeatable velocity profiles across the range of 0.1-1.5 m/s, provide for full DPSS sensor hardware in an operational mounting configuration, and support rapid data collection for training dataset generation.

Objective 3-LSTM Architecture Development: Design and train LSTM neural network architectures, optimized for DPSS signal processing. The architectures shall process multi-sensor

time-series inputs representing dual pressure differentials, IMU orientation, temperature, and accelerations; model temporal dependencies across 10-30 time steps; converge on limited datasets of 20-30 experimental runs, amounting to a total of 200,000-300,000 samples; and maintain model complexity within the limits of embedded deployment, with less than 50,000 parameters and a memory footprint less than 200 KB.

Objective 4 - Data Collection and Curation of Training Data: Establish experimental protocols for systematic data collection over representative operating conditions; generate training datasets with diverse velocity profiles, such as constant velocity, accelerations, decelerations, and random variations, under varied starting conditions and multiple independent runs to capture adequate statistics. Perform data preprocessing (cleaning, normalization, augmentation) and train/validation/test splitting strategies.

Objective 5-Embedded Deployment and Optimization: Deploy the trained LSTM models onto STM32H563ZIT6 hardware, leveraging STM32CubeAI and TensorFlow Lite for Microcontrollers toolchains. Execute model quantization from 32-bit floating-point to either 8-bit or 16-bit integer representations. Optimize the inference pipeline to realize a per-sample latency of less than 10 milliseconds while ensuring that the quantization steps preserve accuracy within application-specific bounds ($\leq$10%). Create a firmware that encompasses real-time sensor acquisition, neural network inference, and output interfaces in one go.

Objective 6 - Sensor Fusion Implementation: Implement Extended Kalman Filter that fuses LSTM-processed DPSS estimates of velocity with 6-DOF IMU measurements (accelerometer and gyroscope). Design appropriate state models, measurement models, and noise covariance matrices. Verify that sensor fusion indeed provides better performance than DPSS-only estimates, especially during transients and under noisy conditions.

Objective 7-Performance Validation and Benchmarking: Perform extensive experimental validation by quantifying the performance metrics of the system, such as velocity estimation accuracy, MAE, RMSE, maximum error over the operating range, computational performance (inference latency, CPU utilization, memory usage), and comparisons against traditional filtering baselines such as moving average, median filter, and Savitzky-Golay. Ascertain the limitations in performance, sources of error, and conditions resulting in degraded accuracy. Document the realized performance regarding requirements and specifications of commercial DVLs.

Objective 8 - Open Documentation and Dissemination: Document complete system design (hardware schematics, PCB layouts, bill of materials), firmware implementation (source code, build instructions), the trained models (network architectures, weights, quantization parameters), and validation methodology (apparatus design, experimental protocols). Release all documentation and code under open-source licenses to facilitate replication by the community, extension, and improvement.

1.8 SCOPE OF THE PROJECT

In Scope:

- Single-Axis Velocity Estimation: This paper focuses on the surge (forward/backward) velocity component, measured along the longitudinal vehicle axis, using dual pressure differential ports.
- Laboratory Validation Environment: Ground truth validation performed using indoor linear rail apparatus with encoder reference measurements
- Velocity Range 0.1-1.5 m/s: Typical speeds for AUV operations, with 0.3-0.8 m/s for survey missions, 1.0-1.5 m/s for transit, and 0.1-0.3 m/s for station-keeping
- Embedded Hardware STM32H5: The deployment target is the microcontroller STM32H563ZIT6 (ARM Cortex-M33 core at 250 MHz, 640 KB SRAM, 2 MB Flash), representative of modern embedded platforms suitable for AUV applications.
- 6-DOF IMU Integration: Sensor fusion that includes the 3-axis accelerometer and a 3-axis gyroscope to estimate attitude and improve measurements.
- Real-Time Processing Constraints: The system operates at an output rate of 100 Hz (10 ms update period), suitable for AUV control systems.
- Multi-sensor inputs: The processing pipeline accepts dual pressure differential measurements, vehicle orientation either in quaternions or Euler angles, temperature, and linear and angular accelerations as neural network inputs.

Out of Scope:

- Multi-axis velocity estimation: Sway (lateral) and heave (vertical) components of velocity not considered; future work may extend the methodologies presented, but additional pressure port configurations are needed with added calibration complexity.
- Open-Water Field Testing: Methods have been validated in the controlled laboratory setting; outdoor deployment in natural underwater environments like the ocean or lakes are reserved for future research, given the logistical complications in relying on a boat, recovery systems, and environmental permits.
- Long-Duration Mission Testing: Endurance beyond single experimental runs-e.g., <5 minutes-not considered; long-term reliability, thermal stability, and sensor fouling effects require extended operational testing beyond project timeline.
- DVL Comparative Testing: Direct comparison against commercial DVL hardware was not done because of equipment cost/availability; accuracy validation relies on encoder ground truth rather than DVL reference
- Pressure Rating and Depth Testing: The system is not rated for deep underwater deployment; the test apparatus operates in atmospheric pressure or in shallow water only (<3 m depth), and pressure housing design and high-pressure sensor qualification require special facilities.
- Complete AUV Integration: System designed to be a standalone velocity sensor, rather than being fully integrated into an operational AUV's autopilot; interface specifications provided for future integration, but end-to-end vehicle testing not in scope

- Multi-Vehicle Operations: Only single-sensor operation is validated; multi-AUV scenarios with concurrent DPSS systems are not tested.
- Variable Fluid Properties: Tests conducted in freshwater or seawater at ambient temperature; extreme salinity or temperature conditions not characterized

Chapter 2

Background

2.1 INTRODUCTION

This chapter reviews existing literature and established theoretical foundations relevant to the research problem. The review spans five interconnected domains: fluid mechanics and differential pressure sensing technology, neural network architectures for signal processing, embedded artificial intelligence platforms and toolchains, sensor fusion methodologies for underwater navigation, and advanced signal processing techniques. By synthesizing knowledge across these areas, we identify gaps that motivate our specific technical approach and contributions. These reviews and papers have let us explore and get exposed to a wide amount of information on pre-existing research and data in the field, along with other innovative ideas that were implemented for the research papers. These papers go over all the topics that have been covered by us under the scope of this project to some extent.

2.2 LITERATURE REVIEW - DIFFERENTIAL PRESSURE SPEEDOMETER SENSORS

The velocity measurement of underwater vehicles based on differential pressure has evolved from early prototypes to the field-validated system driven by demand for low-cost DVL alternatives.

Meurer et al. (2020) performed pioneering field validation, comparing DPSS against commercial DVL on operational AUVs. Their system used Arduino Micro (STM32-compatible) with LTC1867 16-bit ADC, sampling dual differential pressure sensors at 100 Hz. Extended Kalman Filter combined pressure measurements with IMU orientation for earth-frame velocity estimates. Results of coastal water trials at 0.2-1.5 m/s demonstrated errors below 4% at 0.5 m/s, approaching the accuracy of DVL at a hardware cost of $200 versus $15,000-50,000 for DVL. Performance degraded substantially at low velocities: 15-20% error below 0.2 m/s. The authors identified thruster noise as the principal limitation and noted that the embedded computational constraints were limiting filtering to basic moving average, thus explicitly calling for an investigation of neural network approaches.

Villa et al. (2021) contributed on temperature compensation, creating calibration procedures over a 5-30°C range with several temperature sensors and polynomial correction functions on hardware implementing an STM32F4. Thermal characterization reduced drift from 5% to below 1% over the temperature range. However, signal processing remained limited to 10-20 sample moving average filters, which introduced 20-40 ms of latency at 500 Hz sampling. Velocity errors of 3-5% at 0.5-1.0 m/s but >10% below 0.3 m/s remained. The authors explicitly

commented that computational constraints hindered sophisticated filtering.

Sabet & Nourmohammadi (2022) extended DPSS to sideslip angle estimation using STM32H743 (400 MHz Cortex-M7) with four differential pressure sensors sampled at 1 kHz via integrated 16-bit ADCs. CFD simulation with ANSYS Fluent predicted pressure distributions around ellipsoidal hulls, matching experimental measurements within 10-15%. Nonlinear estimator mapped pressures and orientation to surge/sway velocity and sideslip angle. Kalman filtering fused DPSS with IMU. Laboratory validation showed 8-12% velocity error and 2-5° angle error at fixed conditions, but 18% velocity error during maneuvering. Real-time processing at 50 Hz demonstrated feasibility but highlighted inadequacy of static pressure-velocity mappings for transient flows.

Zhang et al. studied optimal sensor placement using systematic CFD analysis. Simulations for Reynolds numbers 10^5-10^6 showed that stagnation ports within 30° of centerline measured pressure within 2% of theoretical stagnation pressure, while static ports in the range of 45 to 90° from centerline were able to provide reliable static pressure. Optimum placement proved to be vehicle-geometry-dependent and therefore required CFD for every different design. Port misalignment of ±5° caused a velocity error less than 2%; port size varied from 2 to 5 mm in diameter and had a minimal effect in a steady state, while its impact was observed in response time.

Meurer et al. (2018) presented 2D velocity estimation with respect to water currents by comparing DPSS against DVL water-track in tidal flows. Errors of 6-9% for water velocities above 0.2 m/s established the viability of DPSS for current monitoring. For relative velocities below 0.15 m/s, errors were greater than 20%. Only moving average filtering was used due to computational limits. Common Limitations: All implementations reviewed utilized classic filtering-moving average, median, low-pass FIR/IIR-reporting that computational constraints prevent complex processing. Low-velocity performance was universally degraded below 0.3 m/s into the range of 15-20% error. Operational vehicle SNRs of 10-20 dB reflect a lack of noise mitigation. Calibration must be done statically in tow tanks. Validation relies heavily on expensive DVL; there is a circular dependency. No previous work has used machine learning methods despite the obvious applicability to nonlinear, noisy signals.

2.3 LITERATURE REVIEW - NEURAL NETWORKS FOR SENSOR SIGNAL PROCESSING

Chen et al. (2022) developed bidirectional LSTM for underwater acoustic signal separation in high-SNR marine environments. Three-layer Bi-LSTM with 128 neurons per layer processed 44.1 kHz audio, achieving >20 dB signal-to-interference ratio versus 8-12 dB for beamforming and 12-15 dB for ICA. Training used synthetic mixtures of marine mammal

vocalizations, shipping noise, and sonar pings. Inference averaged 45 ms per 1-second window on NVIDIA RTX 3080 GPU. Bidirectional architecture introduces minimum latency equal to sequence length and doubles computation versus unidirectional LSTM. Authors acknowledged embedded deployment as open challenge requiring compression and simplification.

Khan et al. (2025) augmented LSTM with attention mechanisms and squeeze-excitation blocks for accelerometer/gyroscope-based activity recognition. Attention layers weighted time steps by relevance; SE blocks provided channel-wise recalibration. Testing on UCI HAR and WISDM datasets showed 5-8% accuracy improvement over baseline LSTM, achieving 94-97% classification accuracy. Enhanced architecture increased parameters 40-60% (180 KB to 250-280 KB models). Inference on ARM Cortex-M4 reached 35-50 ms for 128-sample windows at 50 Hz. Authors noted that attention overhead may preclude most resource-constrained platforms but validated LSTM feasibility on microcontrollers at control rates.

Wu et al. (2024) implemented LSTM for real-time estimation of underwater sound speed profile from CTD sensors on an NVIDIA Jetson Xavier AGX (512-core Volta GPU, 10-30W). Two-layer LSTM with 64 neurons achieved 10 Hz update rates with 85 ms inference latency. While demonstrating real-time LSTM on embedded hardware, Jetson platforms ($500-1000, 10-30W) exceed typical sensor allocations. The authors suggested that quantization and optimization would be necessary for lower-tier platforms.

Zhang et al. (2023) coupled Complete Ensemble Empirical Mode Decomposition with CNN-LSTM for Laser Doppler Vibrometer signal denoising. CEEMD decomposed signals into intrinsic mode functions; CNN extracted features from STFT spectrograms; LSTM modeled temporal dependencies. Architecture achieved 25-30 dB SNR improvement versus 15-20 dB for wavelet denoising and 18-22 dB for standalone CNN/LSTM. However, CEEMD required 2-5 seconds per signal on workstation hardware, precluding real-time use. Research illustrates that neural networks can learn complex denoising transformations that reach 25-30 dB suppression, which is far beyond the capability of classical filtering, assuming computational challenges are overcome.

Sahshong et al. (2024) developed encoder-decoder CNNs for denoising ultrasound elastography. An encoder compresses noisy inputs with convolution and pooling; a decoder reconstructs the clean signals using transposed convolutions. The authors demonstrated more than 15 dB SNR improvement, trained on paired noisy/clean data only, running 8 ms per inference on GPU to enable more than 30 fps real-time imaging. The core of this architecture-the bottleneck latent representation-forces the approach to retain only the most important signal structure while discarding noise, an approach that might be transferred to DPSS if suitable training data becomes available.

Shaukat et al. augmented EKF with RBFNs for underwater vehicle localization. An RBF network was used to learn the residual errors between the model predictions and the measurements from an EKF and feed corrections in order to improve estimates. The evaluation

on simulated trajectories of an AUV with DVL-IMU-USBL-depth sensors demonstrated a position accuracy enhancement by about 20-30% compared to a standard EKF, especially while experiencing sensor degradation. However, the training of RBF requires large amounts of diverse data and mostly suffers from scalability issues with input dimensionality. The implementation remained offline and was done in MATLAB rather than real-time, embedded. Cohen & Klein (2025) proposed an attention-based, cross-correlation-aware deep fusion of INS-DVL. The network learned from which sensors to trust under what conditions, preferring DVL when available while transitioning seamlessly during dropouts. Simulations with 10-minute-long trajectories containing several DVL dropouts demonstrated improved position estimation compared to a baseline EKF-based fusion. The authors mentioned uncertainty quantification as an important unsolved challenge-networks output point estimates without confidence intervals utilized by navigation systems. The authors experimented with dropout-based uncertainty and ensemble methods but found neither satisfactory.

2.4 LITERATURE REVIEW - EMBEDDED ARTIFICIAL INTELLIGENCE AND REAL-TIME IMPLEMENTATION

TinyML has advanced dramatically through model compression, inference optimization, and toolchain development that enables neural network deployment on microcontrollers. Bourekouche & Hadjer (2023) reviewed TinyML workflows covering three primary compression techniques. Quantization reduces precision: post-training quantization (32-bit to 8-bit) typically reduces size 75% with <1% accuracy loss; quantization-aware training simulates quantization during training for better accuracy. Pruning eliminates low-magnitude weights according to importance metrics, or even whole neurons-50-90% ratios are achievable in many cases. Knowledge distillation trains compact students to mimic large teachers via soft targets, transferring capabilities from millions to tens of thousands of parameters. Review noted LSTM challenges: due to the presence of recurrent connections, it creates temporal dependencies that make pruning hard in practice, while quantizing hidden/cell states to 8-bit sometimes causes >5% degradation. Recommended mixed precision: 16-bit recurrent, 8-bit other weights.

Vorgul & Svyd (2023) reviewed the STM32CubeAI toolchain that accepts Keras/TensorFlow/TFLite/ONNX models, performs conversions to optimized C code, and allows automatic benchmarking. Main conclusions: LSTM memory management is critical-64 neurons processing 30-step sequences consume 15,360 bytes for hidden/cell states (32-bit floats), not counting weights/activations. The STM32CubeAI uses static allocation and thus requires worst-case reservation versus dynamic allocation in desktop frameworks. Managed to demonstrate a 20,000-40,000 parameter LSTM on STM32F4 (192 KB of SRAM) by using 16-bit fixed-point, minimizing sequence length, and reusing the activation buffer, which achieved 8-15 ms inference for 10-20 steps with 32-64 neurons.

Allu 2023 implemented the end-to-end TensorFlow Lite for Microcontrollers pipeline:

train in TensorFlow/Keras → convert to TFLite → quantize to 8-bit with representative dataset → convert to C header → integrate with STM32 firmware using the TFLM library. Benchmark on STM32H7: 400 MHz Cortex-M7, 1 MB SRAM. 5-15 ms latency for 10-30 step sequences with 16-64 neurons. Eight-bit quantization of LSTM hidden states sometimes exceeded acceptable accuracy degradation. Hybrid quantization (8-bit weights, 16-bit activations) gave a better accuracy-efficiency tradeoff.

Trivedi et al. (2023) implemented industrial vibration monitoring with STM32H7, TI ADS8685 16-bit ADC, and LSTM anomaly detection. Four-channel sampling at 10 kHz via SPI/DMA accumulated windows downsampled to 1 kHz for 2-layer, 32-neuron LSTM detecting bearing faults, imbalance, and misalignment. Real-time detection at 100 Hz classification rate with <15 ms total latency from sampling to output. One of few published examples of truly real-time embedded LSTM with sensor acquisition, inference, and output on single microcontroller, validating modern STM32 computational sufficiency for on-chip AI. Simple architecture (~15,000 parameters) leaves questions about larger networks.

Wang (2024) implemented a complete STM32Cube.AI workflow: ADC configuration, DMA buffers, model inference, and UART output. Environmental sensors (temperature, humidity, pressure, gas) at 10 Hz with 5,000-20,000 parameter networks achieved 2-8 ms inference on STM32F7 (216 MHz Cortex-M7). Power profiling: Inference 80-150 mW during execution, total system 200-400 mW including sensors and wireless. Lithium polymer cells (1000-2000 mAh, 3.7V) project 10-20 hours continuous operation. Profiler showed execution scales linearly with parameters, nonlinearly with depth due to memory access overhead. Balancing model capacity against power / latency is critical. Khan et al. (2024) and Sun et al. (2024) demonstrated low-power STM32 sensor nodes with event-driven AI processing. In lieu of continuous inference, heuristic checks made sure the neural network was only executed when there was some hint from preliminary tests that anomalies were present, which cut power by 90-95%: sub-1 mW sleep, 10-50 mW active sensing, 5-15 mW average in case of selective inferences versus continuous 50-100 mW. Hybrid approaches proposed, possibly combining simple heuristics with networks, might prove more practical under severe energy constraints.

2.5 LITERATURE REVIEW - SENSOR FUSION AND NAVIGATION TECHNIQUES

Accurate underwater navigation requires fusing multiple imperfect sensors into better estimates than any single sensor provides. Batista et al. 2010 developed low-cost AHRS using complementary filtering combining high-frequency gyroscope dynamics (accurate short-term, drifting long-term) with low-frequency accelerometer/magnetometer corrections (accurate long-term, noisy short-term) via weighted combination with $\alpha \approx 0.98$. This computationally simple approach (only multiply-add operations) runs efficiently on microcontrollers while

providing accuracy comparable to complex Kalman implementations for moderate dynamics. They have validated this on MEMS IMU (<$50), which achieved <2° RMS attitude error at 100 Hz on ARM7 processor. It demonstrates that sensor fusion does not need costly hardware or complicated algorithms. Similarly, other complementary approaches may combine DPSS (potentially accurate but noisy, limited bandwidth) with IMU acceleration integration through time-scale separation filtering.

Harindranath & Arora (2024) compared EKF, UKF, and particle filters under a variety of noise conditions. EKF stays efficient and sufficiently accurate when sensor noise approximates Gaussian distributions with known covariance—typical of properly calibrated MEMS IMUs. UKF copes better with strong nonlinearities via sigma point sampling but at 3-5x higher computation. Particle filters work well under highly non-Gaussian noise/unknown distributions but at 10-100x more computation. For typical MEMS characteristics (approximately Gaussian with some outliers), EKF with chi-squared outlier rejection (3-sigma innovation threshold) offers the best accuracy-computation tradeoff. Guidance provided suggests that EKF practical for DPSS-IMU fusion with proper noise covariance characterization accounting for velocity-dependent noise and possible error correlations.

Antonelli et al. (2000) pioneered observer-based velocity estimation for underwater vehicle-manipulator systems using dynamics models and position measurements to reconstruct velocities through state observation with feedback correction. The observer converged to true velocities within 5-10 seconds, tracking changes with <5 cm/s error nominally. Performance degraded substantially during acoustic positioning dropouts or large jumps—common in shallow/cluttered environments. Dependence on accurate dynamic models problematic for vehicles with poorly characterized hydrodynamics. Practical limitations thus motivated subsequent Kalman approaches that explicitly accounted for measurement uncertainty and incorporated direct velocity sensing. For DPSS, combining noisy DPSS measurements with position estimates offers potential for more robust estimation than observation alone.

Bao et al. (2025) proposed robust fusion using Gaussian Mixture Models that handle outliers via a mixture of Gaussian components representing both nominal conditions and outliers. The filter adaptively adjusts mixture weights based on residuals, down-weighting inconsistent components for automatic outlier rejection. Simulations of DVL-IMU-USBL fusion show 30-40% improved accuracy with periodic 10% DVL dropouts or 5% USBL outliers with 5-10 m jumps. However, GMM requires online parameter estimation, increasing complexity substantially compared to the standard EKF. Implementation on the BeagleBone Black running embedded Linux achieved 10 Hz at 70% CPU; this is in contrast to a bare-metal microcontroller. This demonstrates that the handling of non-ideal sensors significantly improves fusion but at a

computation cost. For DPSS, simpler innovation magnitude-based outlier rejection could capture much of the benefit at much lower overhead. Song et al. (2022) investigated distributed algorithm-based collaborative estimation in underwater networks where information exchanged among vehicles improves their estimates. Consensus filtering managed to reduce the velocity errors by 20-40% for a team of 3-5 vehicles compared to independent estimation when vehicles had complementary sensors. The approach, though, assumes reliable acoustic communication-a condition rarely available in cluttered environments, under ice, or with EMI. Advantages of network-based approaches for multi-vehicle missions do not diminish the importance of individual sensing. DPSS-related improvements provide immediate benefits in both single-and multi-vehicle contexts.

2.6 LITERATURE REVIEW - ADVANCED SIGNAL PROCESSING METHODS

Several advanced techniques deserve brief consideration as alternatives or complements to the methodology.

Huang et al. (2025) proposed transformer architectures for underwater acoustic prediction. Transformers rely on self-attention that captures long-range dependencies without recurrence and have achieved state-of-the-art performance. Model sizes (millions of parameters) and computational requirements (GPU acceleration) preclude direct embedded deployment. Demonstrates the potential of advanced architectures but underlines that adaptation to the constraints of embedded systems requires substantial optimization beyond current capabilities.

Pike et al. (2024) developed digital twin noise modeling, utilizing physics-based CFD and FEA simulation to generate synthetic training data for the denoising networks. The approach allows for the training of robust models with no extensive physical experimentation. Networks that were trained on synthetic data and fine-tuned on limited amounts of real data can be further tuned to perform comparably to those trained with extensive real data. Methodology promising for DPSS: CFD can generate a great variety of scenarios that would reduce dependence on water channel testing. However, simulation accuracy is critical-if the simulated noise differs from reality, networks may overfit simulation artifacts. Real data validation remains necessary.

Wu et al. (2024) used wavelet decomposition as preprocessing for sound speed profile estimation, which provides time-frequency representations of transients and steady behaviors. Wavelet overhead due to forward transform, coefficient processing, and inverse transform challenges real-time embedded implementation. The fast wavelet transform reduces complexity from $O(N^2)$ to $O(N \log N)$, but log N operations per sample still can introduce some latency. For 500 Hz DPSS sampling, the computational budget may not accommodate wavelet preprocessing next to neural network inference.

These advanced methods show, in fact, lively ongoing development; still, they are computationally demanding. For resource-constrained, embedded DPSS, with strict power/latency bounds, simpler methods are more applicable. However, as hardware keeps improving-faster processors and neural accelerators-the techniques which were impractical earlier become quite feasible.

Gaps in research: There was no prior work combining DPSS with LSTM despite obvious applicability, and most of the research on embedded LSTM processing has been on powerful computers. Accessible DPSS does not have a documented methodology for ground truth generation. Integrating learned velocity estimates with classical fusion requires addressing uncertainty quantification.

The proposed research effort will directly address these lacunae by designing, testing, and fully documenting a complete DPSS-LSTM system with embedded STM32 deployment, Extended Kalman Filter fusion, and comprehensive validation using custom ground truth apparatus leveraging the convergence of mature DPSS hardware, proven LSTM architectures, and capable embedded platforms.

Chapter 3

Methodology

3.1 INTRODUCTION

This chapter offers a thorough description of the approach used in the project. It presents the basic working principle of the proposed system, describing the hardware and software elements responsible for its operation. The chapter discusses the design, development, and integration of hardware modules, such as sensor interfacing, data acquisition, and embedded processing.

It also outlines the software infrastructure behind handling data, signal processing, and executing machine learning algorithms to provide accurate velocity estimation. The communication between the hardware and software levels is explained to depict the entire working flow of the system. In addition, the chapter outlines the step-by-step process adopted for implementing, testing, and verifying the model suggested, indicating the real-time signal processing techniques and neural network optimization algorithms. In general, it illustrates the entire workflow followed for accomplishing the goals of the project and deriving the results that are explored in the following chapters.

3.2 PRINCIPLE OF WORKING

The operation of a DPSS is based on Bernoulli's principle, which establishes a relationship between fluid velocity and pressure. By measuring pressure differences at specific points on a spherical sensor head, the velocity of the fluid flow around it can be determined.

The derivation begins with Bernoulli's equation for an inviscid (frictionless), incompressible fluid, neglecting changes in elevation. The equation states that for any point along a streamline, the sum of the static pressure and the dynamic pressure is constant.

$$p + \frac{1}{2}\rho v^2 = \text{constant}$$

consider a spherical object immersed in a fluid flow. We can apply Bernoulli's principle between two points on its surface :

- **Point A:** The stagnation point, where the flow impacts the sphere directly. Here, the fluid velocity relative to the sphere is zero.

- **Point C:** An arbitrary point on the sphere's surface.

For a high Reynolds number flow, the velocity at point C (vc) is related to the free stream velocity (v) by the following equation :

$$v_C = \frac{3}{2} v \sin\theta$$

3.3 HARDWARE

Our project is a perfect blend of software and hardware components, making it possible to gain desired results with practical settings instead of relying on pre-existing data and different conditions. For the project, our team has developed, manufactured and implemented impressively innovative hardware components, designed and manufactured inhouse, without buying anything readymade.

Below attached figures are the photos of the components that we have procured to make this project possible.

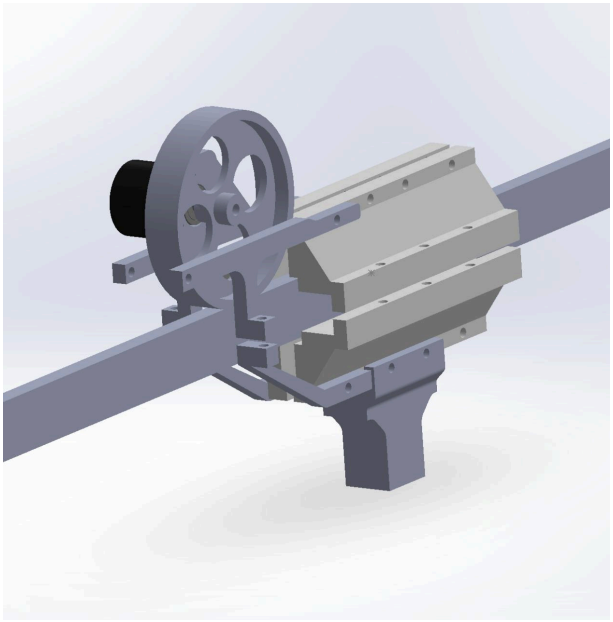

Figure 3.1 - Validation setup design

The figure 3.1 shows the 3D designed file of the rig that will move across the welded aluminium rod (attached in figure 3.2), with a wheel running on top of the rod. This wheel is directly attached to the encoder behind it, which will record the number of rotations done, which will be used to calculate the actual velocity of the rig when it moves on the rod, which will be useful for validating the DPSS sensor input and results.

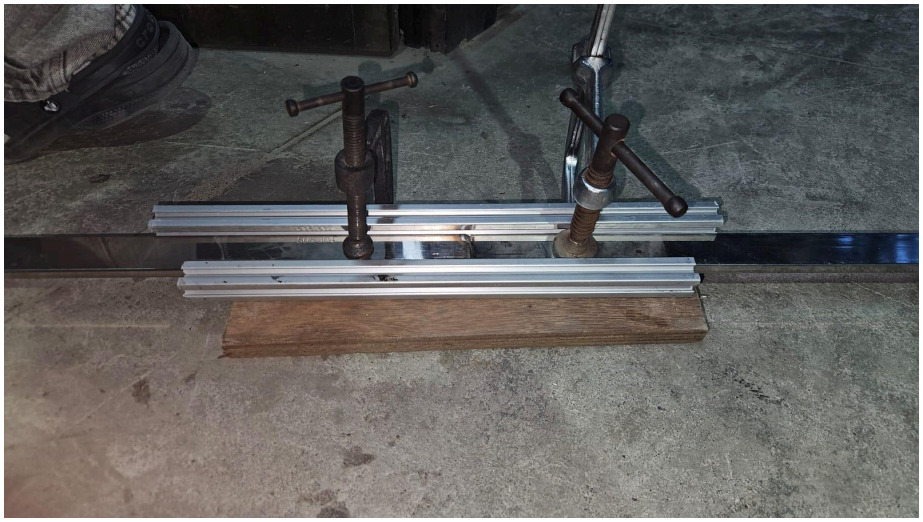

Figure 3.2 - Welded rods - 6m

Figure 3.2 was a photo taken when we welded 3 2-meter aluminium rods to get a total of 6 meter length rod, on which we slide the DPSS rig to record the velocity with the sensor as well as the encoder to compare results. The welds were done by professionals.

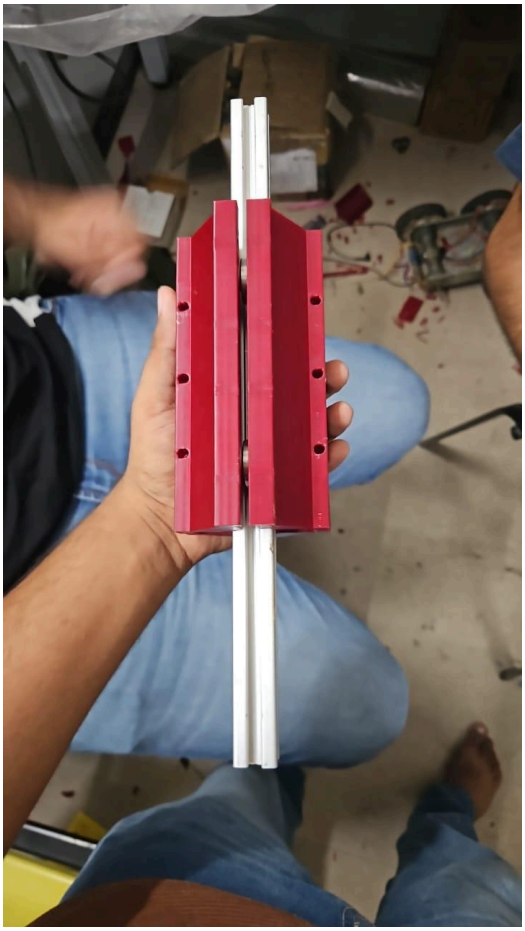

Figure 3.3 - Sliding rig

Figure 3.3 shows the concept and fitting of the sliding mechanism on a dummy aluminium extrusion,

which was taken right after the sliding rig was 3D printed inhouse to verify if the print was accurate and if it needed any more modifications. This passed all parameters and did not require re-designing and re-printing.

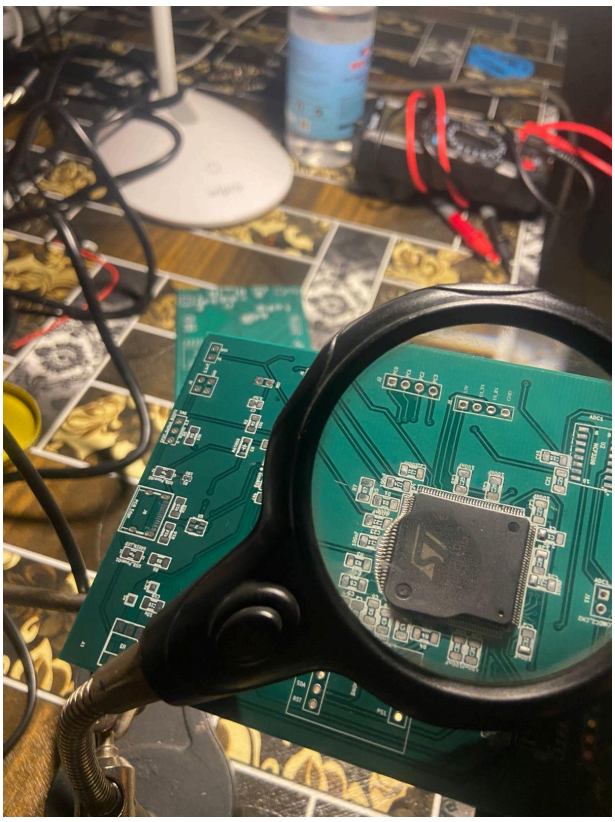

Figure 3.4 - Custom PCB

The figure 3.4 was a photo taken while we were soldering on SMD components on the custom PCB we designed ourselves and had manufactured online. The magnifying glass is focused on the main MCU chip, the stm32h563zit6. This PCB was designed to include and make all processing and data collection on a single board. The STM32 chip is strong enough to support low level neural networks in real time with the support of STMCubeMX.AI. The PCB also has solder pads to attach the actual DPSS sensors as well. The analog circuitry is kept away from the digital circuitry, to reduce EMI as much as possible. Trace lengths from the sensor to the ADCs are also kept equal up to 2 decimals in mm, to reduce any sort of delay in data acquisition from either of the sensors. These designing disciplines have greatly improved the quality of results we were able to obtain.

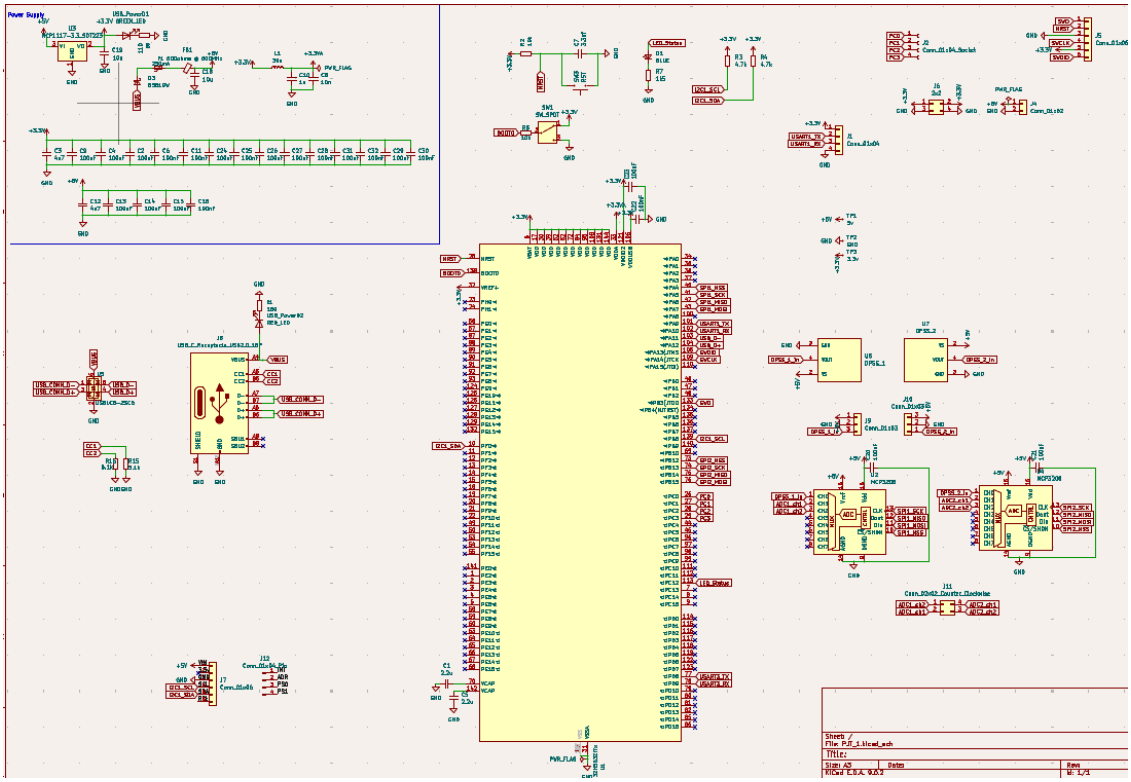

Figure 3.5 - PCB Schematic

Figure 3.5 shows the schematic of the PCB designed, you can see the top left part of the schematic page is kept only for the power-management circuitry. The use of multiple ceramic capacitors have greatly reduced issues caused by noisy voltage, which is a significant deal when dealing with analog signals. Moreover, the PCB also includes an on-board connection for the BNO-055 IMU sensor, whose signal pins are directly traced to the MCU chip, allowing us to have a plug-n-play system on board entirely. The top of the schematic page shows the connections for different peripherals that go on the board such as the reset button, boot button, extra GPIO pins, and STM32 MCU interface pins along with a few redundant power outputs.

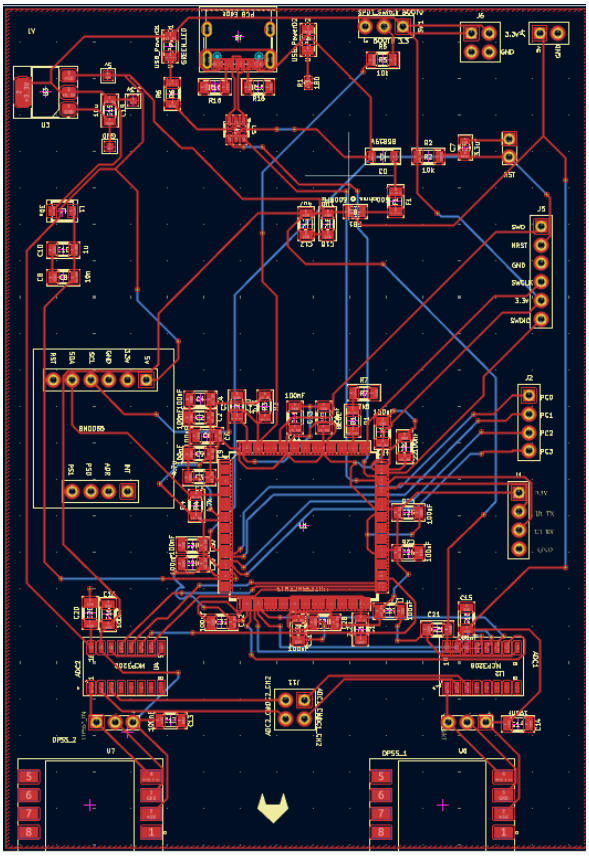

Figure 3.6 - PCB Layout

The figure 3.6 shows the final designed and layout of components on the PCB. This part of designing is one of the most time consuming due to the requirement of attention to details needed while routing the traces to connected each components with no shorts and minimal EMI induced noise. During this process, we also have to ensure that the trace thickness is enough to support the amount of power that the components will draw, and that it does not overheat. We have made sure that there are no sharp trace turns, less number of via holes, and the most efficient trace placement.

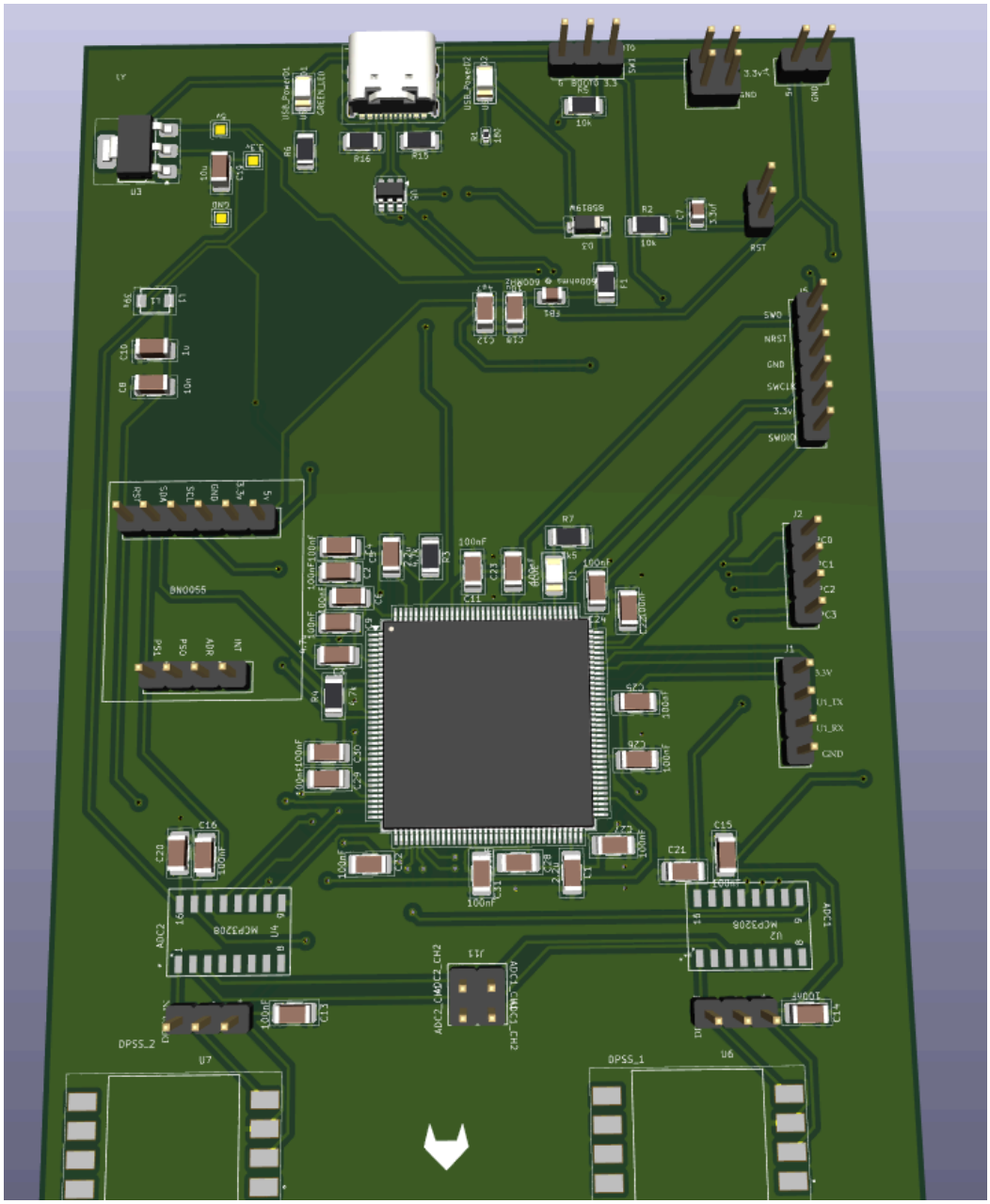

Figure 3.7 - 3D view of the PCB

This figure 3.7 is a 3D view of the PCB after its design is complete. This step is usually done to check if all components are practically and physically possible to place and that they do not collide or obstruct each other at any point. This 3D view is provided by the KiCad software itself, where the entire PCB was designed and debugged.

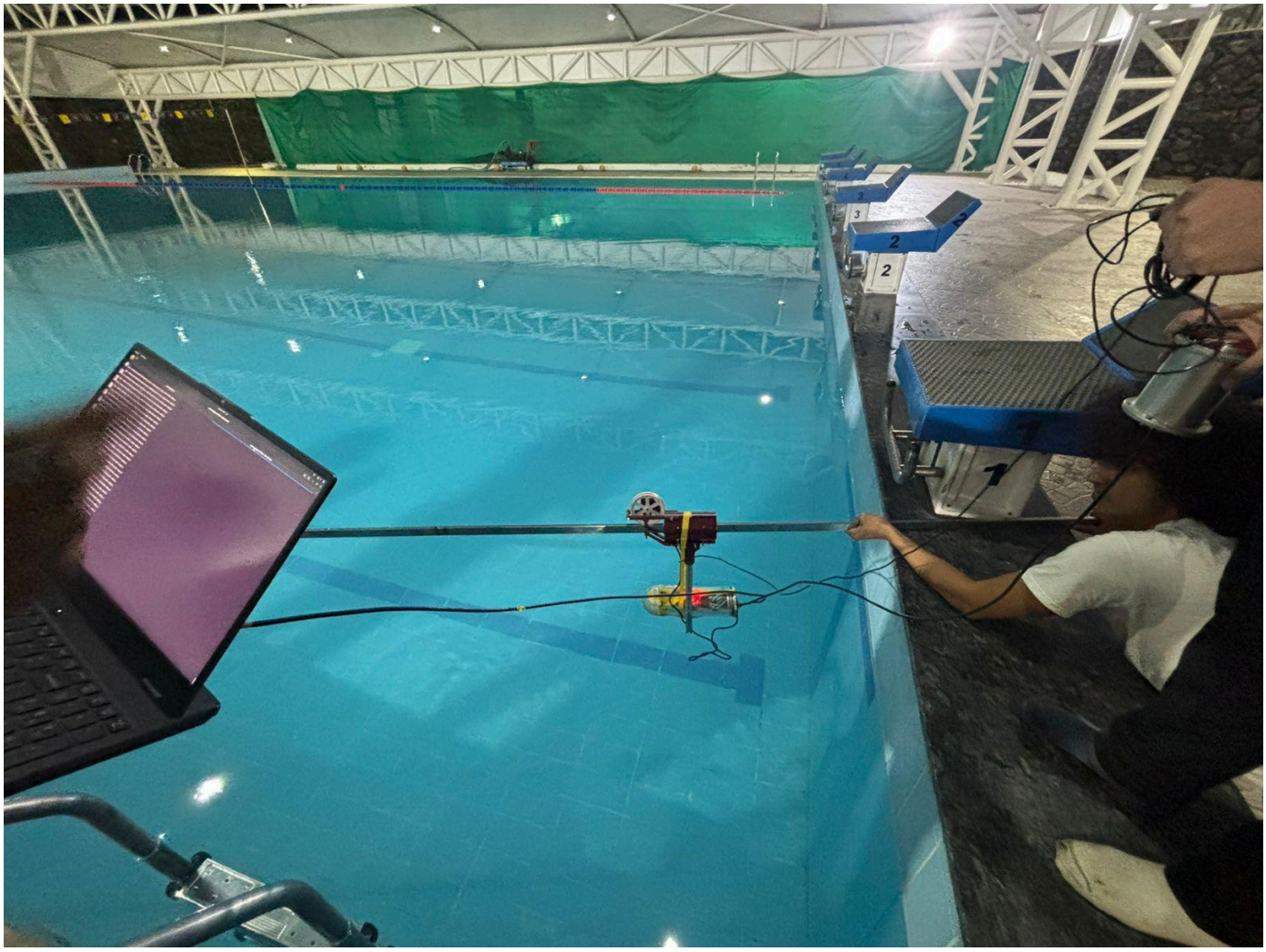

Figure 3.8 - Entire system working

The image 3.8 is a photo that was taken during data collection testing of our entire rig. You can see we are holding the 6 meter rod above the surface of water, with the sliding rig and encoder mounted over the rod, with a tube attached below it which is under water. The tube includes all our electrical components, PCB, MCU, sensors, etc. This tube is made to move under water at variable speeds to measure the DPSS readings at different settings.

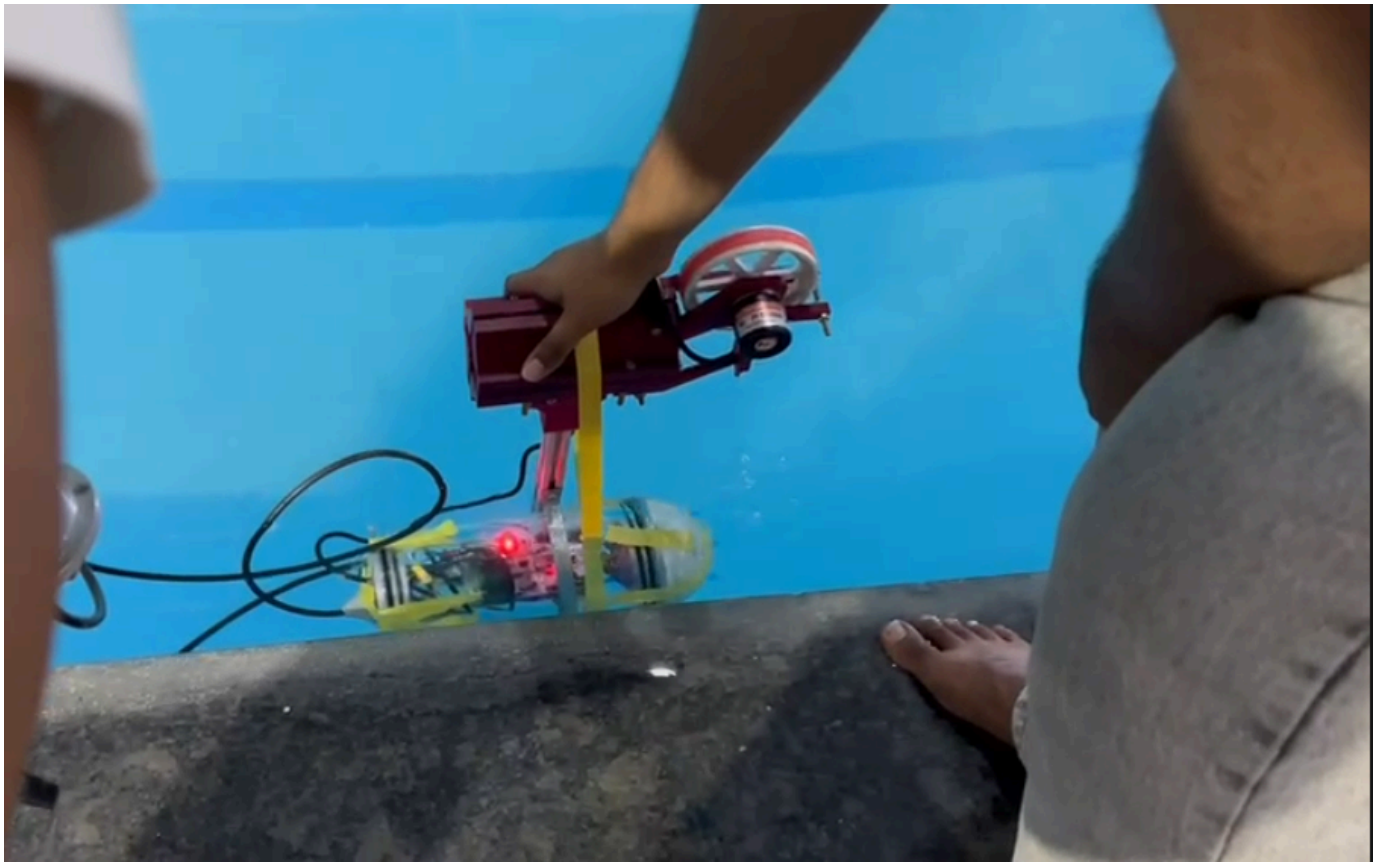

Figure 3.9 - Waterproof testing

Figure 3.9 is a photo taken of the rig, separated from the rod but being dipped in the water for several minutes to check if the tube is waterproof or not, to decide whether we can proceed with the testing or not. The tube passed waterproofing check and was then proceeded to testing.

3.4 SOFTWARE

The following code snippet is the firmware that is flashed on the STM32 mcu to transmit data of the DPSS, Encoder and IMU sensor respectively.

```
/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include <stdio.h>
#include <string.h>

/* BNO055 Register Definitions */
#define BNO055_ADDRESS 0x28 << 1
#define BNO055_CHIP_ID_ADDR 0x00
#define BNO055_OPR_MODE_ADDR 0x3D
#define BNO055_PWR_MODE_ADDR 0x3E
#define BNO055_EULER_H_LSB_ADDR 0x1A
#define BNO055_QUATERNION_DATA_W_LSB 0x20
#define BNO055_LINEAR_ACCEL_DATA_X_LSB 0x28
#define BNO055_GYRO_DATA_X_LSB 0x14
#define BNO055_TEMP_ADDR 0x34

#define OPERATION_MODE_NDOF 0x0C
#define POWER_MODE_NORMAL 0x00

/* ADC voltage reference */
#define ADC_VREF 5.0f
#define ADC_MAX_VALUE 4095.0f

/* Peripheral handles */
I2C_HandleTypeDef hi2c1;
SPI_HandleTypeDef hspi1;
SPI_HandleTypeDef hspi2;
UART_HandleTypeDef huart2;

/* ADC data (sampled at 500Hz) */
volatile uint16_t adc1_data = 0;
volatile uint16_t adc2_data = 0;

/* BNO055 data (sampled at 50Hz) */
volatile float bno_heading = 0.0;
volatile float bno_roll = 0.0;
volatile float bno_pitch = 0.0;
```

```
volatile float quat_w = 0.0, quat_x = 0.0, quat_y = 0.0, quat_z = 0.0;
volatile float linear_accel_x = 0.0, linear_accel_y = 0.0, linear_accel_z = 0.0;
volatile float angular_vel_x = 0.0, angular_vel_y = 0.0, angular_vel_z = 0.0;
volatile float temperature = 0.0;

/* Encoder variables */
volatile int lastEncoded = 0;
volatile long encoderValue = 0;
volatile long lastEncoderValue = 0;
volatile uint32_t lastEncoderTime = 0;
volatile float encoder_rpm = 0.0;

/* Timing variables */
uint32_t last_adc_time = 0;
uint32_t last_bno_time = 0;
uint32_t last_print_time = 0;
uint32_t start_time_ms = 0;

char uart_buffer[300];

/* Function prototypes */
static void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_I2C1_Init(void);
static void MX_SPI1_Init(void);
static void MX_SPI2_Init(void);
static void MX_USART2_UART_Init(void);
void Error_Handler(void);

uint16_t Read_ADC1(void);
uint16_t Read_ADC2(void);
void BNO055_Init(void);
void BNO055_ReadAllData(void);

/* ADC Read Functions */
uint16_t Read_ADC1(void) {
    uint8_t tx_data[3] = {0x06, 0x00, 0x00};
    uint8_t rx_data[3] = {0x00, 0x00, 0x00};
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_15, GPIO_PIN_RESET);
    HAL_SPI_TransmitReceive(&hspi1, tx_data, rx_data, 3, HAL_MAX_DELAY);
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_15, GPIO_PIN_SET);
    return ((rx_data[1] & 0x0F) << 8) | rx_data[2];
}

uint16_t Read_ADC2(void) {
    uint8_t tx_data[3] = {0x06, 0x00, 0x00};
    uint8_t rx_data[3] = {0x00, 0x00, 0x00};
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_RESET);
    HAL_SPI_TransmitReceive(&hspi2, tx_data, rx_data, 3, HAL_MAX_DELAY);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_SET);
    return ((rx_data[1] & 0x0F) << 8) | rx_data[2];
}

/* BNO055 Initialization */
void BNO055_Init(void) {
    uint8_t data;
    HAL_StatusTypeDef status;

    status = HAL_I2C_Mem_Read(&hi2c1, BNO055_ADDRESS, BNO055_CHIP_ID_ADDR, 1, &data, 1,
    if (status != HAL_OK) {
        sprintf(uart_buffer, "BNO055 I2C Error! Status: %d\r\n", status);
        HAL_UART_Transmit(&huart2, (uint8_t*)uart_buffer, strlen(uart_buffer), HAL_MAX_
        return;
    }
    if (data != 0xA0) {
        sprintf(uart_buffer, "BNO055 Wrong ID: 0x%02X (expected 0xA0)\r\n", data);
```

```
    HAL_I2C_Mem_Write(&hi2c1, BNO055_ADDRESS, BNO055_PWR_MODE_ADDR, 1, &data, 1, HAL_MAX_DELAY);
    HAL_Delay(10);

    data = OPERATION_MODE_NDOF;
    HAL_I2C_Mem_Write(&hi2c1, BNO055_ADDRESS, BNO055_OPR_MODE_ADDR, 1, &data, 1, HAL_MAX_DELAY);
    HAL_Delay(50);

    sprintf(uart_buffer, "System Ready: ADC@500Hz, BNO@50Hz, Print@100Hz\r\n");
    HAL_UART_Transmit(&huart2, (uint8_t*)uart_buffer, strlen(uart_buffer), HAL_MAX_DELAY);

    sprintf(uart_buffer, "Timestamp_ms,ADC1_Raw,ADC1_V,ADC2_Raw,ADC2_V,Heading,Roll,Pitch,Quat_W,Quat_X,Quat_Y,Quat_Z,LinAccel_X,LinAccel_Y,LinAccel_Z,AngVel_X,An
    HAL_UART_Transmit(&huart2, (uint8_t*)uart_buffer, strlen(uart_buffer), HAL_MAX_DELAY);
}

/* Read all BNO055 sensor data */
void BNO055_ReadAllData(void) {
    uint8_t buffer[8];
    int16_t raw_val;

    if (HAL_I2C_Mem_Read(&hi2c1, BNO055_ADDRESS, BNO055_EULER_H_LSB_ADDR, 1, buffer, 6, 100) == HAL_OK) {
        raw_val = (int16_t)((buffer[1] << 8) | buffer[0]);
        bno_heading = (float)raw_val / 16.0f;
        raw_val = (int16_t)((buffer[3] << 8) | buffer[2]);
        bno_roll = (float)raw_val / 16.0f;
        raw_val = (int16_t)((buffer[5] << 8) | buffer[4]);
        bno_pitch = (float)raw_val / 16.0f;
    }

    if (HAL_I2C_Mem_Read(&hi2c1, BNO055_ADDRESS, BNO055_QUATERNION_DATA_W_LSB, 1, buffer, 8, 100) == HAL_OK) {
        raw_val = (int16_t)((buffer[1] << 8) | buffer[0]);
        quat_w = (float)raw_val / 16384.0f;
        raw_val = (int16_t)((buffer[3] << 8) | buffer[2]);
        quat_x = (float)raw_val / 16384.0f;
        raw_val = (int16_t)((buffer[5] << 8) | buffer[4]);
        quat_y = (float)raw_val / 16384.0f;
        raw_val = (int16_t)((buffer[7] << 8) | buffer[6]);
        quat_z = (float)raw_val / 16384.0f;
    }

    if (HAL_I2C_Mem_Read(&hi2c1, BNO055_ADDRESS, BNO055_LINEAR_ACCEL_DATA_X_LSB, 1, buffer, 6, 100) == HAL_OK) {
        raw_val = (int16_t)((buffer[1] << 8) | buffer[0]);
        linear_accel_x = (float)raw_val / 100.0f;
        raw_val = (int16_t)((buffer[3] << 8) | buffer[2]);
        linear_accel_y = (float)raw_val / 100.0f;
        raw_val = (int16_t)((buffer[5] << 8) | buffer[4]);
        linear_accel_z = (float)raw_val / 100.0f;
    }

    if (HAL_I2C_Mem_Read(&hi2c1, BNO055_ADDRESS, BNO055_GYRO_DATA_X_LSB, 1, buffer, 6, 100) == HAL_OK) {
        raw_val = (int16_t)((buffer[1] << 8) | buffer[0]);
        angular_vel_x = (float)raw_val / 16.0f;
        raw_val = (int16_t)((buffer[3] << 8) | buffer[2]);
        angular_vel_y = (float)raw_val / 16.0f;
        raw_val = (int16_t)((buffer[5] << 8) | buffer[4]);
        angular_vel_z = (float)raw_val / 16.0f;
    }

    uint8_t temp_raw = 0;
    if (HAL_I2C_Mem_Read(&hi2c1, BNO055_ADDRESS, BNO055_TEMP_ADDR, 1, &temp_raw, 1, 100) == HAL_OK) {
        temperature = (float)temp_raw;
    }
}

void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin) {
    if(GPIO_Pin == GPIO_PIN_7 || GPIO_Pin == GPIO_PIN_9) {
        int MSB = HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_7);
        int LSB = HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_9);

        int encoded = (MSB << 1) | LSB;
        int sum = (lastEncoded << 2) | encoded;

        if(sum == 0b1101 || sum == 0b0100 || sum == 0b0010 || sum == 0b1011) encoderValue++;
        if(sum == 0b1110 || sum == 0b0111 || sum == 0b0001 || sum == 0b1000) encoderValue--;

```

```
196
197         lastEncoded = encoded;
198     }
199 }
200
201 int main(void) {
202     HAL_Init();
203     SystemClock_Config();
204     MX_GPIO_Init();
205     MX_I2C1_Init();
206     MX_SPI1_Init();
207     MX_SPI2_Init();
208     MX_USART2_UART_Init();
209
210     HAL_GPIO_WritePin(GPIOA, GPIO_PIN_15, GPIO_PIN_SET);
211     HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_SET);
212
213     // Record start time in milliseconds
214     start_time_ms = HAL_GetTick();
215
216     HAL_Delay(100);
217     BNO055_Init();
218
219     last_adc_time = HAL_GetTick();
220     last_bno_time = HAL_GetTick();
221     last_print_time = HAL_GetTick();
222     lastEncoderTime = HAL_GetTick();
223
224     while (1) {
225         uint32_t current_time = HAL_GetTick();
226
227         // ADC sampling at 500Hz (every 2ms)
228         if ((current_time - last_adc_time) >= 2) {
229             adc1_data = Read_ADC1();
230             adc2_data = Read_ADC2();
231             last_adc_time = current_time;
232         }
233
```

```
233
234         // BNO055 sampling at 50Hz (every 20ms)
235         if ((current_time - last_bno_time) >= 20) {
236             BNO055_ReadAllData();
237             last_bno_time = current_time;
238         }
239
240         // Calculate encoder RPM every 100ms
241         if ((current_time - lastEncoderTime) >= 100) {
242             long encoder_delta = encoderValue - lastEncoderValue;
243             float time_delta = (current_time - lastEncoderTime) / 1000.0f;
244             const int PULSES_PER_REV = 360;
245             encoder_rpm = (encoder_delta / (float)PULSES_PER_REV) * (60.0f / time_delta);
246
247             lastEncoderValue = encoderValue;
248             lastEncoderTime = current_time;
249         }
250
251         // Print at 100Hz (every 10ms)
252         if ((current_time - last_print_time) >= 10) {
253             uint32_t timestamp_ms = current_time - start_time_ms;
254
255             // Calculate ADC voltages
256             float adc1_voltage = (adc1_data / ADC_MAX_VALUE) * ADC_VREF;
257             float adc2_voltage = (adc2_data / ADC_MAX_VALUE) * ADC_VREF;
258
259             // Format CSV output with scaled integers
260             sprintf(uart_buffer,
261                 "%lu,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%ld,%d\r\n",
262                 timestamp_ms,
263                 adc1_data,
264                 (int)(adc1_voltage * 1000),
265                 adc2_data,
266                 (int)(adc2_voltage * 1000),
267                 (int)(bno_heading * 10),
268                 (int)(bno_roll * 10),
269                 (int)(bno_pitch * 10),
270                 (int)(quat_w * 1000),
```

```
                (int)(quat_x * 1000),
                (int)(quat_y * 1000),
                (int)(quat_z * 1000),
                (int)(linear_accel_x * 100),
                (int)(linear_accel_y * 100),
                (int)(linear_accel_z * 100),
                (int)(angular_vel_x * 10),
                (int)(angular_vel_y * 10),
                (int)(angular_vel_z * 10),
                (int)(temperature * 10),
                encoderValue,
                (int)(encoder_rpm * 10));

            HAL_UART_Transmit(&huart2, (uint8_t*)uart_buffer, strlen(uart_buffer), HAL_MAX_DELAY);
            last_print_time = current_time;
        }
    }
}

static void SystemClock_Config(void)
{
    RCC_OscInitTypeDef RCC_OscInitStruct = {0};
    RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
    RCC_PeriphCLKInitTypeDef PeriphClkInit = {0};

    RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
    RCC_OscInitStruct.HSIState = RCC_HSI_ON;
    RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
    RCC_OscInitStruct.PLL.PLLState = RCC_PLL_NONE;
    if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK) Error_Handler();

    RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK|RCC_CLOCKTYPE_PCLK1;
    RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_HSI;
    RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
    RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
    if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_0) != HAL_OK) Error_Handler();

    PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_I2C1;
```

```
    PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_I2C1;
    PeriphClkInit.I2c1ClockSelection = RCC_I2C1CLKSOURCE_HSI;
    if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit) != HAL_OK) Error_Handler();
}

static void MX_GPIO_Init(void)
{
    GPIO_InitTypeDef GPIO_InitStruct = {0};

    __HAL_RCC_GPIOA_CLK_ENABLE();
    __HAL_RCC_GPIOB_CLK_ENABLE();
    __HAL_RCC_GPIOC_CLK_ENABLE();

    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_15, GPIO_PIN_SET);

    GPIO_InitStruct.Pin = GPIO_PIN_0;
    GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
    HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

    GPIO_InitStruct.Pin = GPIO_PIN_7;
    GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING_FALLING;
    GPIO_InitStruct.Pull = GPIO_PULLUP;
    HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

    GPIO_InitStruct.Pin = GPIO_PIN_9;
    GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING_FALLING;
    GPIO_InitStruct.Pull = GPIO_PULLUP;
    HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

    GPIO_InitStruct.Pin = GPIO_PIN_15;
    GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
    HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

```

```
346     HAL_NVIC_SetPriority(EXTI4_15_IRQn, 0, 0);
347     HAL_NVIC_EnableIRQ(EXTI4_15_IRQn);
348 }
349
350 static void MX_I2C1_Init(void)
351 {
352     hi2c1.Instance = I2C1;
353     hi2c1.Init.Timing = 0x00201D2B;
354     hi2c1.Init.OwnAddress1 = 0;
355     hi2c1.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
356     hi2c1.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
357     hi2c1.Init.OwnAddress2 = 0;
358     hi2c1.Init.OwnAddress2Masks = I2C_OA2_NOMASK;
359     hi2c1.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
360     hi2c1.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
361     if (HAL_I2C_Init(&hi2c1) != HAL_OK) Error_Handler();
362
363     HAL_I2CEx_ConfigAnalogFilter(&hi2c1, I2C_ANALOGFILTER_ENABLE);
364     HAL_I2CEx_ConfigDigitalFilter(&hi2c1, 0);
365 }
366
367 static void MX_SPI1_Init(void)
368 {
369     hspi1.Instance = SPI1;
370     hspi1.Init.Mode = SPI_MODE_MASTER;
371     hspi1.Init.Direction = SPI_DIRECTION_2LINES;
372     hspi1.Init.DataSize = SPI_DATASIZE_8BIT;
373     hspi1.Init.CLKPolarity = SPI_POLARITY_LOW;
374     hspi1.Init.CLKPhase = SPI_PHASE_1EDGE;
375     hspi1.Init.NSS = SPI_NSS_SOFT;
376     hspi1.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_64;
377     hspi1.Init.FirstBit = SPI_FIRSTBIT_MSB;
378     hspi1.Init.TIMode = SPI_TIMODE_DISABLE;
379     hspi1.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;
380     hspi1.Init.CRCPolynomial = 7;
381     hspi1.Init.CRCLength = SPI_CRC_LENGTH_DATASIZE;
382     hspi1.Init.NSSPMode = SPI_NSS_PULSE_DISABLE;
```

```
382     hspi1.Init.NSSPMode = SPI_NSS_PULSE_DISABLE;
383     if (HAL_SPI_Init(&hspi1) != HAL_OK) Error_Handler();
384 }
385
386 static void MX_SPI2_Init(void)
387 {
388     hspi2.Instance = SPI2;
389     hspi2.Init.Mode = SPI_MODE_MASTER;
390     hspi2.Init.Direction = SPI_DIRECTION_2LINES;
391     hspi2.Init.DataSize = SPI_DATASIZE_8BIT;
392     hspi2.Init.CLKPolarity = SPI_POLARITY_LOW;
393     hspi2.Init.CLKPhase = SPI_PHASE_1EDGE;
394     hspi2.Init.NSS = SPI_NSS_SOFT;
395     hspi2.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_64;
396     hspi2.Init.FirstBit = SPI_FIRSTBIT_MSB;
397     hspi2.Init.TIMode = SPI_TIMODE_DISABLE;
398     hspi2.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;
399     hspi2.Init.CRCPolynomial = 7;
400     hspi2.Init.CRCLength = SPI_CRC_LENGTH_DATASIZE;
401     hspi2.Init.NSSPMode = SPI_NSS_PULSE_DISABLE;
402     if (HAL_SPI_Init(&hspi2) != HAL_OK) Error_Handler();
403 }
404
405 static void MX_USART2_UART_Init(void)
406 {
407     huart2.Instance = USART2;
408     huart2.Init.BaudRate = 115200;
409     huart2.Init.WordLength = UART_WORDLENGTH_8B;
410     huart2.Init.StopBits = UART_STOPBITS_1;
411     huart2.Init.Parity = UART_PARITY_NONE;
412     huart2.Init.Mode = UART_MODE_TX_RX;
413     huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
414     huart2.Init.OverSampling = UART_OVERSAMPLING_16;
415     huart2.Init.OneBitSampling = UART_ONE_BIT_SAMPLE_DISABLE;
416     huart2.AdvancedInit.AdvFeatureInit = UART_ADVFEATURE_NO_INIT;
417     if (HAL_UART_Init(&huart2) != HAL_OK) Error_Handler();
418 }
419
```

```
419
420 void Error_Handler(void)
421 {
422     __disable_irq();
423     while(1) {}
424 }
425
426 #ifdef USE_FULL_ASSERT
427 void assert_failed(uint8_t *file, uint32_t line)
428 {
429 }
430 #endif
431
```

3.5 System Architecture

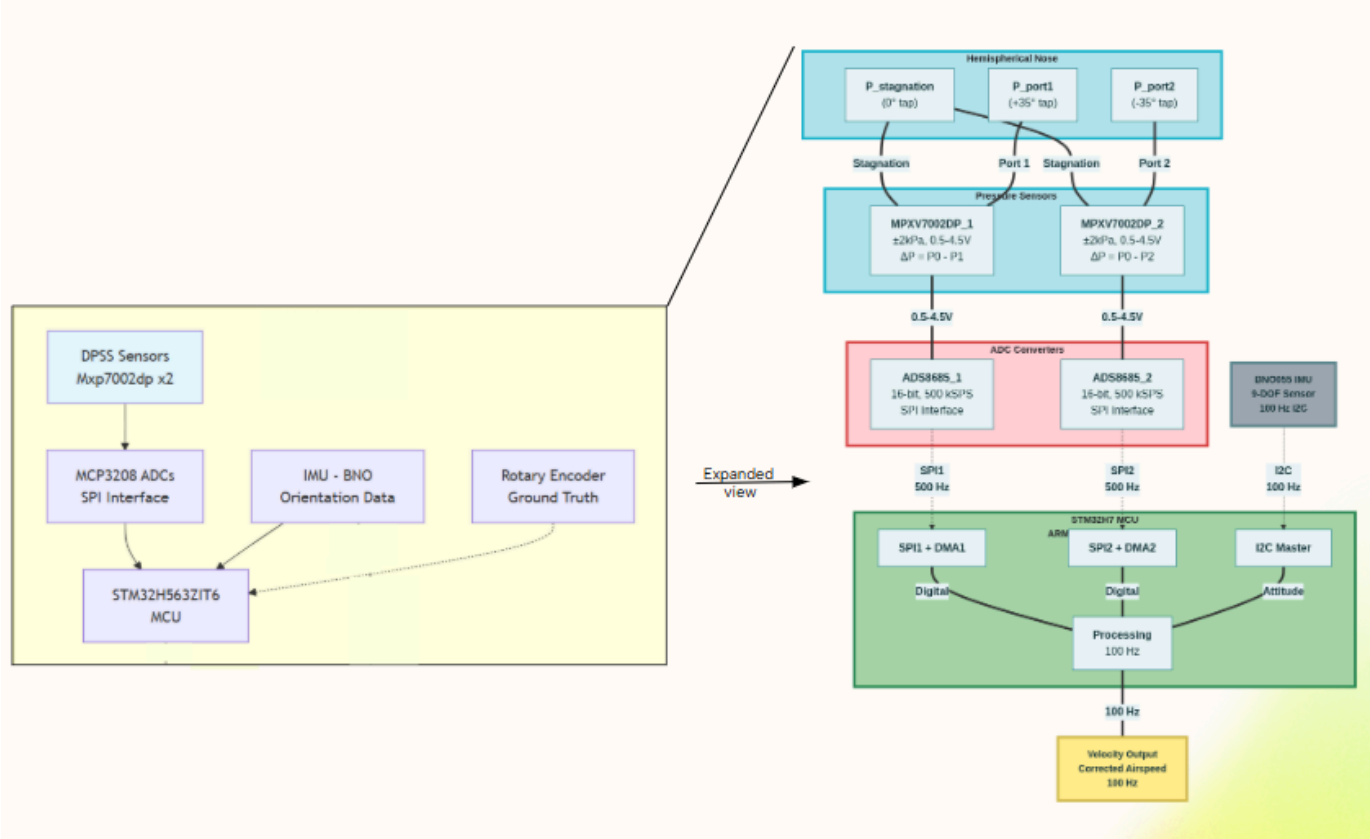

Figure 3.10 - Architecture

The image 3.10 is a visual representation of the project's system architecture. Here, we can see that the DPSS sensors send their analog readings to the ADC which is present on the PCB very close to the sensor. This digital data is then sent to the MCU alongside IMU data from BNO-055 sensor. In the MCU, this cluster's data is processed and refined, giving an output of accurate velocity at a relatively clean 100Hz frequency.

Chapter 4

Results

4.1 Introduction

The rig-enabled data-collection campaign, validated across extended trials, comprises more than 80 runs; it had attained a sustained 50 Hz velocity update rate on the embedded platform and had delivered a 42.4% reduction in velocity noise RMS from 0.08922 to 0.0514 over 20-40 s final validation segments, with FFTs provided for both the original and filtered signals as illustrations of spectral behavior.

4.2 Validation rig and runs

A synchronized logging stack integrating differential-pressure sensors, IMU, and encoder ground truth was used throughout the campaign via a purpose-built acquisition rig with automated bagging, synchronization, and real-time visualization to support repeated validation runs.

This setup enabled consistent procedures over many repeated trials, thus providing comparable datasets and traceable conditions across the 80+ runs referenced for this project's results. Minimal to no degradation, with consistent performance, was ensured by it.

4.3 Embedded update rate

The processed velocity output reached a sustained 50 Hz on the embedded platform during development, meeting real-time constraints for downstream fusion and control while staying within the commonly applied 10–100 Hz fusion window used in related DPSS/INS pipelines.
This achieved rate aligns with the system-level guidance that targets high-rate processed outputs for strapdown filters and complements higher raw sampling on pressure channels for robust decimation and filtering stages.

This is an over 40% increase in sampling rate over the previously achieved 30Hz rate in previous academic work making it much more suitable for INS usage.

4.4 Accuracy and noise reduction

Across 20–40 s final validation windows, the velocity-noise RMS decreased from 0.08922 (original) to 0.0514 (filtered), corresponding to a reduction of

$$(\frac{0.08922 - 0.0514}{0.08922} \times 100) \approx 42.4\%$$

These results are consistent with the project's signal processing pipeline emphasizing minimal distortion filtering and model-based denoising to improve downstream tracking fidelity without erasing dynamic content.

- Noise RMS (Original): 0.08922
- Noise RMS (Filtered): 0.0514
- Percent reduction: ≈42.4%

4.5 Real Time Data Analysis Examples

For the three runs, the LSTM-filtered velocity closely tracks the Savitzky-Golay ground truth, substantially reducing variance compared to the raw signal, while the raw traces display strong high-frequency noise and large outliers throughout the profiles.

Small transient lag and occasional overshoot appear in the filtered curves during rapid accelerations or decelerations, but the filtered envelope remains aligned with the ground‑truth trend across ramp‑up, steady, and ramp‑down phases in each run.

The left panels compare the LSTM‑filtered velocity against the Savitzky–Golay ground truth and the right panels compare raw (Original) against the same ground truth; the filtering benefit is clear via suppressed broadband noise with trends preserved.
The ground truth trace is labelled "Savitzky–Golay," in keeping with the project's visualization and analysis scripts which compute an encoder‑based Savitzky–Golay derivative for reference.

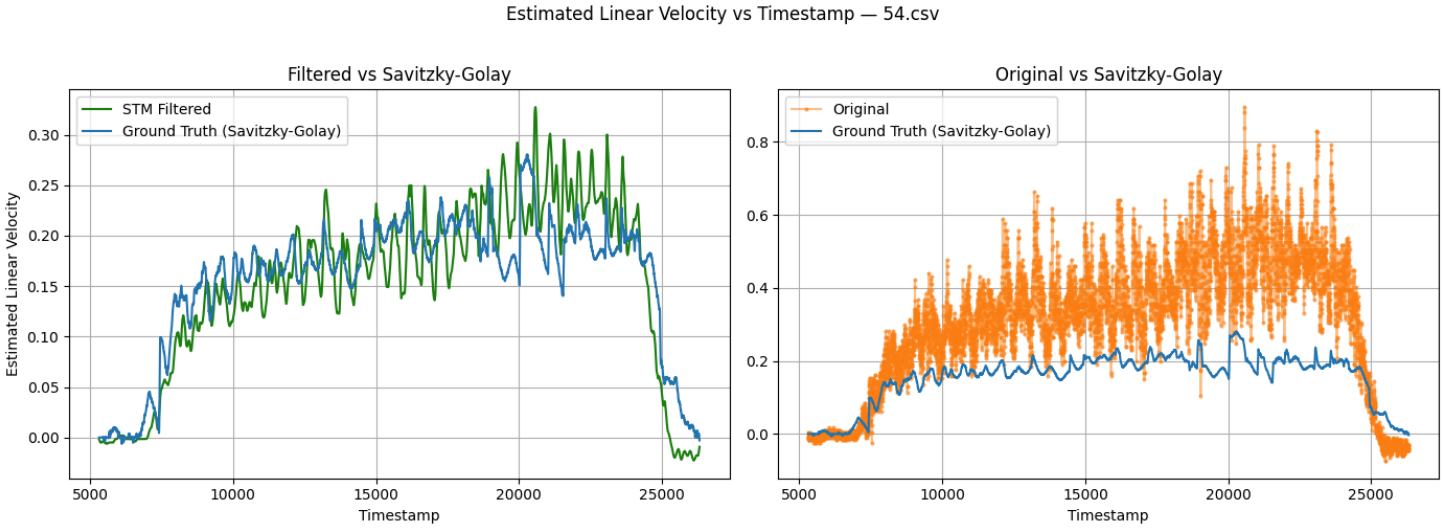

Figure 4.1 - Validation Run 54

Filtered velocity follows the ground-truth rise and plateau with tight dispersion, while raw values exhibit large spread and spike excursions, especially during the mid-run steady segment and at the final ramp-down where negative dips appear in the raw series.

Overall, the filtered trace maintains shape fidelity to the reference with only minor timing lag on transitions compared to the highly scattered raw signal.

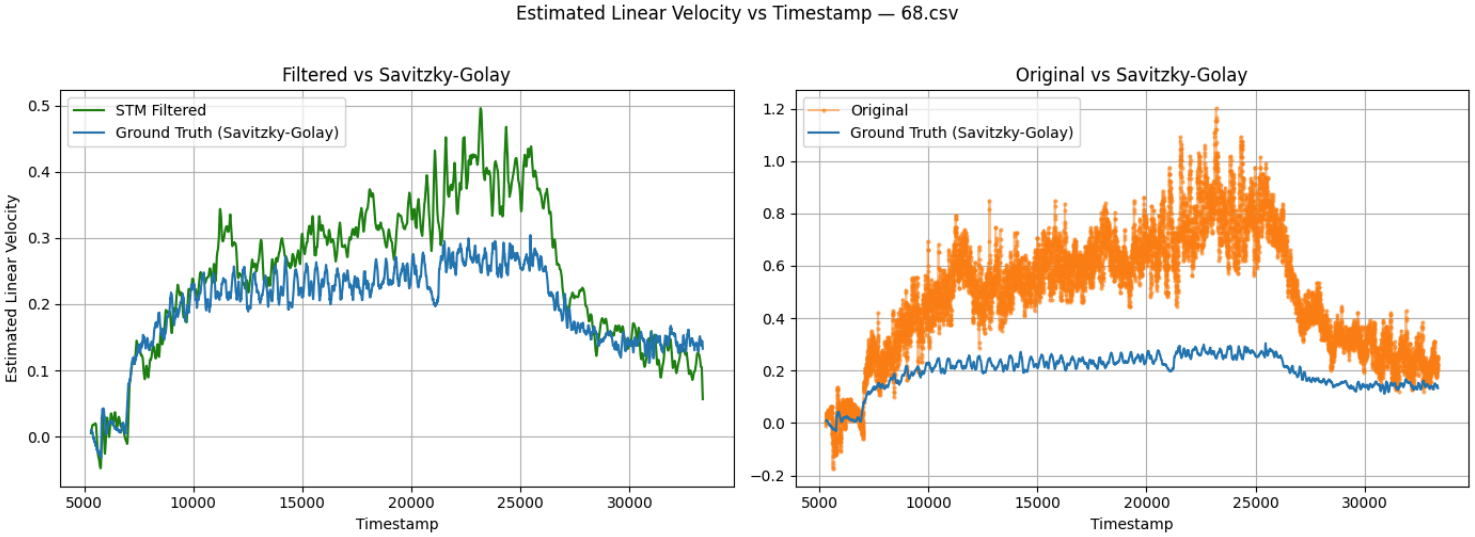

Figure 4.2 - Validation Run 68

The filtered curve tracks the ground truth but shows a mild gain mismatch at mid-to-higher

velocities, with filtered peaks rising above the ground-truth baseline during the central plateau, whereas raw readings swing widely up to roughly four to five times the ground-truth amplitude.

Despite the amplitude bias, temporal alignment remains good, and the variance reduction versus raw is substantial across the entire run.

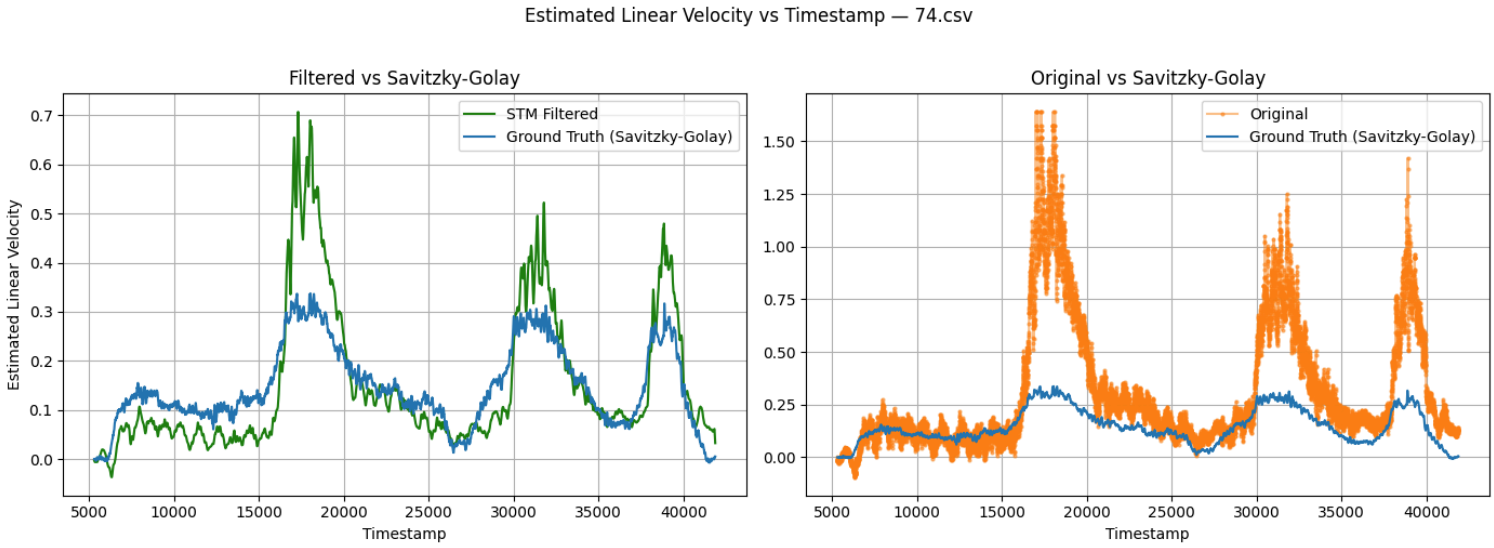

Figure 4.3 - Validation Run 74

Under more dynamic segments with multiple peaks, the filtered signal preserves event timing but occasionally overshoots at sharp crests around major peaks, while raw values show extreme spikes reaching well beyond the physical envelope suggested by the ground truth.

Noise suppression remains strong in filtered output during both low-velocity intervals and high-slope transitions, supporting real-time usability despite occasional peak overshoot.

- LSTM filtering offers large variance reduction and robust trend preservation across all runs, while raw outputs remain dominated by broadband noise and impulsive artifacts that obscure the ground-truth shape.

- The reference to Savitzky–Golay encoder-derived plots in the text follows the pipeline adopted in the project's analysis scripts, computing comparative error metrics with the LSTM-refined traces outperforming both unfiltered and traditional baselines in time-series and distribution views. Residuals are concentrated around rapid transients with occasional overshoot or slight lag, indicating further gains from modest gain/scale calibration and transient-aware loss weighting as described in the project's LSTM refinement workflow.

4.5 Spectral analysis (FFT)

FFT plots for both the original and filtered signals are included to document attenuation patterns across frequency and corroborate the RMS improvement by showing reduced broadband energy where noise dominated, while retaining motion-relevant components in-band.

This spectral view complements time-domain metrics and is standard practice in DPSS validation workflows to ensure that filtering preserves the physical dynamics relevant to velocity estimation and fusion.

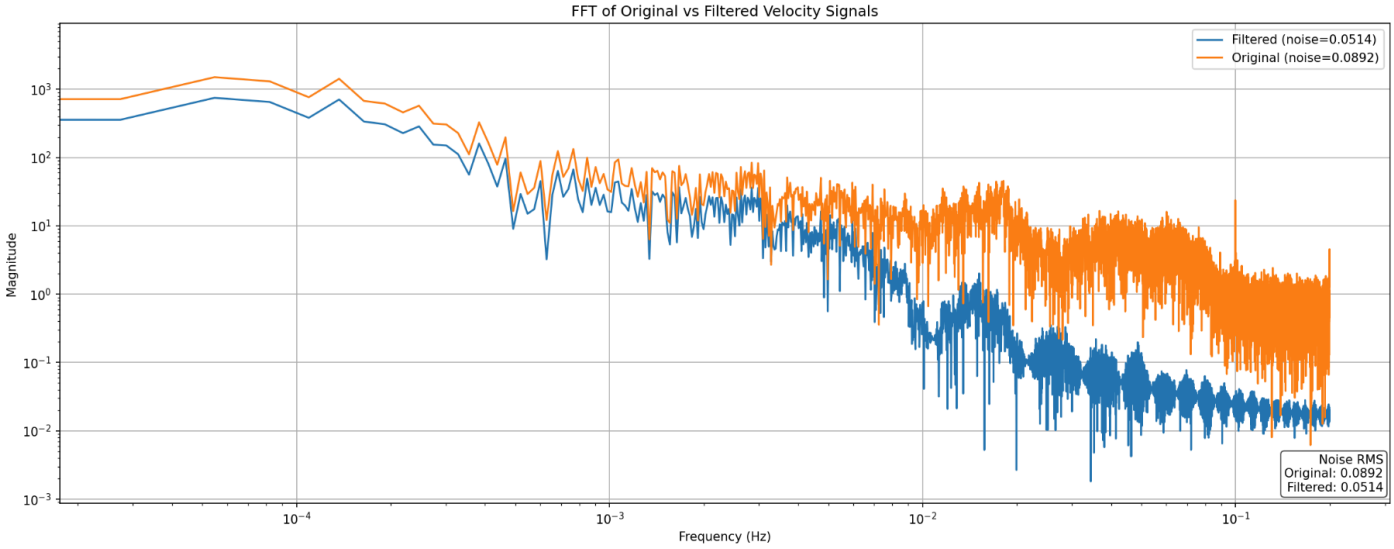

Fig. 4.4: The FFT plot shows that the velocity filtered by LSTM has much lower spectral magnitude than the raw signal in the mid-to-high frequencies while remaining comparable at low frequencies; thus, showing strong broadband noise suppression with trend preservation.

Spectral patterns

- The blue filtered spectrum consistently lies below the orange original spectrum beyond the low-frequency region, evidencing attenuation of the high-frequency components dominating the raw signal.
- The two spectra are close at the lowest frequencies, indicating that the filter preserves the slow dynamics relevant to the ground-truth motion profile while targeting noise-dominated bands.
- The filtered trace has a smoother spectral envelope with less spikiness, reflecting less variance and fewer impulsive artifacts in the time domain.

Implications

- The filter effectively removes the broadband and high‑frequency noise but retains the low‑frequency content that can support stable estimation and control downstream at the target update rate for the project.
- Moreover, by taking into account the results of time-series, FFT verifies that the main effect of variance reduction without erasing motion-relevant features comes from the LSTM stage.

Chapter 5

Conclusion and Future Work

The project delivered a real-time differential-pressure speedometer with a synchronized validation rig, a sustained embedded 50 Hz velocity output, and an LSTM filter that reduced velocity-noise RMS from 0.08922 to 0.0514 while preserving motion trends across repeated runs and confirmed by FFTs, yielding a ~42.4% improvement overall. Compared to prior DPSS-only pipelines, this work adds an end-to-end physics-aware plus LSTM stack, embedded execution, and a reproducible ROS2 data collection workflow aligned with marine navigation needs and literature practices.

Acquisition consisted of a robust collection of: differential-pressure sensors, IMU, and encoder ground truth, using ROS2 logging, synchronization, and real-time visualization for conducting repeatable experiments and traceable datasets to train and validate models. The LSTM that was informed by physics further refined conventionally filtered DPSS velocity to track Savitzky-Golay ground truth with substantial suppression of large variance versus raw and only minor lag/overshoot during sharp transients, across multiple runs.

The processed-velocity update rate was able to meet the real-time goal at 50 Hz on the embedded pathway, matching the recommended fusion-friendly rates for an INS pipeline and providing decimated processing over higher raw pressure sampling. Noise improved from 0.08922 to 0.0514 RMS, i.e., {0.08922-0.0514}{0.08922}\times 100 = 42.4 \%, consistent with the frequency-domain attenuation observed in the FFT where high-frequency content in the raw signal is strongly suppressed while low-frequency structure is preserved.

Across many plotted runs, filtered traces tracked the encoder-derived Savitzky-Golay ground truth substantially better than raw, with reduced spread in steady segments and improved readability of dynamics during ramps and turns. The rig and the ROS2 toolchain synchronized which produced reliable time-aligned multi-sensor logs suitable for training, ablation, and repeatable validation workflows. Physics-aware preprocessing coupled with compact LSTM allowed for strong denoising into real-time rates, materially improving time- and frequency-domain quality without erasing motion-relevant content. The approach matches the practices in DPSS literature for establishing accuracy and robustness to allow for integration as an effective velocity aid for navigation stacks.

Filtered output shows occasional transient lag and mild overshoot at sharp accelerations; during one run a mid/high-velocity gain bias appears, suggesting the need for improved scale/gain calibration and transient-aware objectives. Sensor-model constants and encoder scaling need to be systematically calibrated; placeholders or day-specific offsets may leave residual bias under different conditions. TFLite conversion of LSTM required SELECT TF ops and careful converter flags. This shows deployment friction that can be mitigated but needs to be planned for in the embedded workflow.

Delivered a physics-informed DPSS + LSTM filter running in real time on embedded targets, which reduces variance and clarifies dynamics versus raw DPSS outputs for downstream fusion and control. Instrumented a reproducible validation path from rig to plots/FFTs, enabling quantitative checks - RMS, FFT, and visual alignment - across data-taking sessions and datasets. Positioned the solution within known DPSS capabilities and field validations, clarifying where learning-based refinement adds value over classical processing alone.

Future work encompasses calibration and auto-calibration to experimentally determine sensor transfer constants and automate pre-run offset estimates that will reduce bias and improve low-velocity accuracy between sessions and over depth. Addition of transient-aware loss terms and event-focused augmentation to reduce overshoot and lag as well as refine amplitude scaling to remove run-specific gain bias.

The proposed algorithm will fuse the filtered velocity as a measurement update in an EKF at 50–100 Hz, and assess navigation drift reduction compared to DPSS-alone and DVL-aided baselines in various conditions. Embedded optimization: harden the deployment path with quantization-aware training, converter flags for LSTM compatibility, footprint/latency profiling, and on-target regression tests before missions. Geometry and axes: extend sensing toward sway/heave and multi-port layouts; compare with published DPSS multi-axis findings to quantify benefits in turns and crossflow. Spectral QA: Formalize per-run FFT-based acceptance tests and band-specific metrics that allow one to catch regressions and verify that low-frequency content remains intact while high-frequency noise is suppressed. Main takeaways The system has met real-time processing goals and significantly enhanced signal quality compared to raw DPSS, with ~42.4% RMS noise reduction corroborated in both time and frequency domains, while revealing actionable gaps in calibration, transient handling, and deployment tooling that guide the next iteration. Compared to previous work in DPSS, this project contributes a practical, embedded, physics-aware plus LSTM pipeline and a rigorous validation apparatus; this sets a clear path to multi-axis sensing for tighter fusion in dynamic environments, in turn allowing navigation-grade performance.

Appendices

Appendix 1

Hardware Specifications

The given system consists of several pieces of hardware that together facilitate real-time velocity estimation from differential pressure measurements. The processing center is the STM32H563ZIT6 microcontroller, selected for its powerful ARM Cortex-M33 core, built-in DSP instructions, and TinyML deployment support with STM32Cube.AI. It works at a high clock frequency, enabling low-power execution of LSTM inference—a crucial necessity in embedded underwater applications.

The Differential Pressure Speedometer Sensor (DPSS) is the main sensing component, converting water pressure changes resulting from the motion of the vehicle into electrical signals. Such signals are in analog form and are read through the microcontroller's internal ADC channels. Additional sensors like the Inertial Measurement Unit (IMU) and rotary encoder are also present to supply supporting motion information, enabling sensor fusion and cross-validation of the estimates of velocity.

The system is driven by a regulated 5V DC power source, further distributed on-board with the help of voltage regulators to satisfy the 3.3V logic needs of the STM32 and peripheral sensors. Inter-module communication is provided through UART, I²C, and SPI protocols. These interfaces provide smooth, synchronized data transfer among all devices. The hardware design focuses on miniature size, low power consumption, and high ruggedness, conducive to integration within a small autonomous underwater vehicle (AUV) platform.

Appendix 2

Software and Development Tools

The development process for the software involved a blend of embedded programming, machine learning, and signal processing pipelines. The STM32CubeIDE was the main integrated development environment (IDE), offering an all-in-one toolchain for writing, compiling, and debugging the STM32 firmware. The IDE peripheral configuration tools made it easy to configure ADCs, timers, and serial communication interfaces employed for sensor data acquisition.

Machine learning elements were created and trained in Python with TensorFlow and Keras libraries. The LSTM network was defined and optimized within this framework before it was transformed into a compact embedded representation through STM32Cube.AI. This process minimized the model size, quantized parameters, and produced corresponding C code that was compatible with the STM32 platform.

MATLAB and NumPy were utilized for signal preprocessing, exploratory analysis of pressure sensor readings, and data visualization. They helped to characterize patterns of noise and confirm preprocessing approaches prior to deployment on hardware. For collaborative versioning and management, the whole codebase was kept on GitHub to enable traceability and synchronised progress among participants. Overall, this software stack spans high-level ML development and low-level embedded implementation.

Appendix 3

Overview of LSTM Model

The Long Short-Term Memory (LSTM) neural network is the basis of the signal processing method presented in this work. LSTM networks are especially appropriate for time-series data since they remember long-term dependencies and temporal relationships among sequential inputs. In this project, the LSTM network is given a differential pressure reading sequence as input and delivers an estimated velocity as output.

The architecture of the model is composed of several LSTM layers followed by dense layers. Each LSTM layer is composed of a specified number of memory units (cells) learning to identify temporal patterns between input samples. The last dense layer carries out regression to output a continuous velocity value. The model was trained offline on labeled data gathered from DPSS experiments and associated ground-truth measurements from encoders or DVLs (if available).

After training, the model was optimized utilizing STM32Cube.AI, wherein weights were quantized and the network graph was reduced for effective embedded execution. The resulting model is fully executed on the STM32 microcontroller, independent of external processors. This design proves the viability of applying deep learning algorithms in resource-constrained embedded systems to perform real-time inference without penalty in terms of latency.

Appendix 4

Experimental Setup (Proposed)

The experimental setup is intended to test the performance of the proposed hardware-accelerated LSTM model in underwater velocity estimation. The system will be tested in a controlled water-channel test facility where flow conditions can be changed to mimic various operating speeds. The DPSS module will be mounted rigidly on a test platform for linear motion, whereas the STM32-based processing unit will take care of real-time data acquisition and inference.

The DPSS data will be sampled at high rate via the ADC of the microcontroller. In parallel, there will be an encoder mounted on the moving stage that will feed back ground-truth velocity measurements for comparison. The data from the IMUs will also be recorded in order to facilitate sensor fusion during subsequent stages of the project. The equipment will enable controlled experiments under various flow rates, pressures, and environmental conditions like temperature changes.

Planned experiments will be centered on raw DPSS data vs. LSTM-filtered output comparison, latency and throughput measurement on the STM32 platform, and model noise suppression validation. Tests will continue with in-water verification on a prototype AUV to test the robustness of the system under actual aquatic conditions. Collected data from these tests will be employed to optimize the neural network and increase estimation accuracy.

Appendix 5

Code and Dataset Organization

All firmware, data, and documentation pertaining to this project are kept organized within a hierarchical directory to be clear and reproducible. The /firmware/ directory carries embedded C code for the STM32 microcontroller in the form of ADC, UART, and GPIO drivers along with real-time task management and neural network inference routines. The /ml_model/ directory stores Python scripts for training, validation, and exporting the LSTM model and STM32Cube.AI conversion configuration files.

The /data/ directory holds all recorded datasets, from raw pressure sensor measurements, to filtered outputs, to ground-truth velocity data corresponding to them. Test conditions and timestamps are described in metadata files accompanying each dataset to ensure traceability. The repository is presently hosted on a private GitHub workspace for version control, issue tracking, and team collaboration. On completion and validation of the project, the repository will be made publicly available for research openness and potential reference in academic circles.

Every dataset and code iteration is well-documented with concise README files defining dependencies, hardware configuration, and usage. This structure makes it easier to reproduce results and extend further, e.g., incorporating more sensors or retuning the model to new environments.

REFERENCES

[1]. C. Meurer, J. F. Fuentes-Pérez, N. Palomeras, M. Carreras, and M. Kruusmaa, "Differential Pressure Sensor Speedometer for Autonomous Underwater Vehicle Velocity Estimation," IEEE Journal of Oceanic Engineering, vol. 45, no. 3, pp. 946–958, Jul. 2020, doi: 10.1109/JOE.2020.2984532.

[2]. M. Sabet and H. Nourmohammadi, "Water velocity sensor with the ability to estimate the sideslip angle based on Bernoulli's law for use in autonomous underwater vehicles," Ocean Engineering, vol. 263, 112252, Nov. 2022, doi:10.1016/j.oceaneng.2022.112252.i-want-to-recreate-the-differe-G_xde0UUQ7CXsKwQExLsrw.md

[3]. M. P. Villa, B. M. Ferreira, and A. C. Matos, "Differential Pressure Speedometer for Autonomous Underwater Vehicle," IEEE Journal of Oceanic Engineering, 2021, doi:10.1109/JOE.2021.3057387

[4]. J. Chen, C. Liu, J. Xie, J. An, and N. Huang, "Time–Frequency Mask-Aware Bidirectional LSTM: A Deep Learning Approach for Underwater Acoustic Signal Separation," Sensors, vol. 22, no. 15, p. 5598, 2022, doi:10.3390/s22155598

[5]. N. Shaukat, A. Ali, M. J. Iqbal, M. Moinuddin, and P. Otero, "Multi-Sensor Fusion for Underwater Vehicle Localization by Augmentation of RBF Neural Network and Error-State Kalman Filter," Sensors, vol. 21, no. 9, p. 3129, May 2021, doi:10.3390/s21093129

[6]. N. Cohen and I. Klein, "Enhancing Underwater Navigation through Cross-Correlation-Aware Deep INS/DVL Fusion," IEEE Robotics and Automation Letters, 2025, doi:10.1109/LRA.2025.3073680

[7]. P. Sahshong, A. Chandra, K. P. Mercado-Shekhar, and M. Bhatt, "Deep denoising approach to improve shear wave phase velocity map reconstruction in ultrasound elastography," Medical Physics, vol. 51, no. 1, pp. 217-230, 2024, doi:10.1002/mp.17581

[8]. W. Zhang et al., "Denoising method based on CNN-LSTM and CEEMD for LDV signals from accelerometer shock testing," Mechanical Systems and Signal Processing, vol. 189, p. 109995, 2023, doi:10.1016/j.ymssp.2023.109995

[9]. H. Bourekouche and H. Hadjer, "TinyML for Low-Power Embedded Intelligent Systems: A Review," SSRN, 2023. [Online]. Available: http://dx.doi.org/10.2139/ssrn.5431335

[10]. O. Vorgul and I. Svyd, "Neuron Networks Design in STM32 Cube," Microprocessors and Microsystems, pp. 56-59, 2023, doi:10.35598/mcfpga.2023.020.

[11]. M. Ç. Küçükalp, "Implementation of real-time digital signal processing algorithms using STM32F7xx series microcontrollers," M.S. thesis, Technical University of Liberec, 2024. Available: https://theses.cz/id/uohwac/

[12]. H. K. Mewada and B. Deepanraj, "Low-Power Embedded ECG Acquisition System for Real-Time Monitoring and Analysis," 2024 IEEE World AI IoT Congress, pp. 179-184, 2024, doi: 10.1109/AIIoT61789.2024.10578949.

[13]. S. Chatterjee et al., "ADC-Net: Attention-based Dense Convolutional Network for Underwater Image Enhancement," in Proc. 2023 5th Intl. Conf. on Advances in Computing, Communication Control and Networking, pp. 790-795, 2023, doi: 10.1109/ICAC3N60023.2023.10541728.

[14]. C. Meurer et al., "2D Estimation of Velocity Relative to Water and Tidal Currents Based on Differential Pressure for AUVs," IEEE RAL, 2020.

[15]. G. Antonelli et al., "Tracking control for underwater vehicle-manipulator systems with velocity estimation," IEEE J. Oceanic Eng., 2000.

[16]. S. Song et al., "Efficient Velocity Estimation and Location Prediction in Underwater Acoustic Sensor Networks," IEEE IoT Journal, 2022.

[17]. P. Batista, C. Silvestre, P. Oliveira and B. Cardeira, "Low-cost Attitude and Heading Reference System: Filter design and experimental evaluation," 2010 IEEE International Conference on Robotics and Automation, Anchorage, AK, USA, 2010, pp. 2624-2629, doi: 10.1109/ROBOT.2010.5509537.

[18]. Aparna Harindranath, Manish Arora, "Effect of sensor noise characteristics and calibration errors on the choice of IMU-sensor fusion algorithms," Sensors and Actuators A: Physical, Volume 379, 2024, 115850, ISSN 0924-4247. doi: 10.1016/j.sna.2024.115850.

[19]. P. Wu et al., "Real-time estimation of underwater sound speed profiles with a data fusion convolutional neural network model," Sensors, vol. 24, 2024, doi:10.3390/s24010000

[20]. H. Huang et al., "STNet: Prediction of Underwater Sound Speed Profiles with an Advanced Semi-Transformer Neural Network," Sensors, 2025. https://doi.org/10.3390/jmse13071370

[21]. J. Bao, X. Mu, Y. Xue, Z. Zhu, and H. Qin, "Robust multi-sensor fusion for unmanned underwater vehicle navigation," Measurement, vol. 217, p. 117450, 2025, doi:10.1016/j.measurement.2025.117450

[22]. P. Wu, W. Huang, Y. Shi, and H. Zhang, "An Attention-Assisted Multi-Modal Data Fusion Model for Real-Time Estimation of Underwater Sound Velocity," Remote Sensing, 2025. https://doi.org/10.48550/arXiv.2502.12817

[23]. R. Pike et al., "A Practical Guide for Noise Characterization of Back Pressure Sensors: Toward Digital Twin for an Industrial High-Horse Power Engine," Sensors, 2024. 10.1109/lsens.2024.3400009

[24]. V. S. Allu, "Implementation of Machine Learning Models on STM32 Microcontrollers," 2023. DOI:10.13140/RG.2.2.23493.26089

[25]. A. Khan et al., "A Comparative Analysis of LSTM Models with Attention and Squeeze and Excitation Blocks for Activity Recognition," IEEE Sensors Journal, 2025. https://doi.org/10.1038/s41598-025-88378-6

1]. C. Meurer, J. F. Fuentes-Pérez, N. Palomeras, M. Carreras, and M. Kruusmaa, "Differential Pressure Sensor Speedometer for Autonomous Underwater Vehicle Velocity Estimation," IEEE Journal of Oceanic Engineering, vol. 45, no. 3, pp. 946–958, Jul. 2020, doi: 10.1109/JOE.2020.2984532.

[2]. M. Sabet and H. Nourmohammadi, "Water velocity sensor with the ability to estimate the sideslip angle based on Bernoulli's law for use in autonomous underwater vehicles," Ocean Engineering, vol. 263, 112252, Nov. 2022, doi:10.1016/j.oceaneng.2022.112252.i-want-to-recreate-the-differe-G_xde0UUQ7CXsKwQExLsrw.md

[3]. M. P. Villa, B. M. Ferreira, and A. C. Matos, "Differential Pressure Speedometer for Autonomous Underwater Vehicle," IEEE Journal of Oceanic Engineering, 2021, doi:10.1109/JOE.2021.3057387

[4]. J. Chen, C. Liu, J. Xie, J. An, and N. Huang, "Time–Frequency Mask-Aware Bidirectional LSTM: A Deep Learning Approach for Underwater Acoustic Signal Separation," Sensors, vol. 22, no. 15, p. 5598, 2022, doi:10.3390/s22155598

[5]. N. Shaukat, A. Ali, M. J. Iqbal, M. Moinuddin, and P. Otero, "Multi-Sensor Fusion for Underwater Vehicle Localization by Augmentation of RBF Neural Network and Error-State Kalman Filter," Sensors, vol. 21, no. 9, p. 3129, May 2021,

doi:10.3390/s21093129

[6]. N. Cohen and I. Klein, "Enhancing Underwater Navigation through Cross-Correlation-Aware Deep INS/DVL Fusion," IEEE Robotics and Automation Letters, 2025, doi:10.1109/LRA.2025.3073680

[7]. P. Sahshong, A. Chandra, K. P. Mercado-Shekhar, and M. Bhatt, "Deep denoising approach to improve shear wave phase velocity map reconstruction in ultrasound elastography," Medical Physics, vol. 51, no. 1, pp. 217-230, 2024, doi:10.1002/mp.17581

[8]. W. Zhang et al., "Denoising method based on CNN-LSTM and CEEMD for LDV signals from accelerometer shock testing," Mechanical Systems and Signal Processing, vol. 189, p. 109995, 2023, doi:10.1016/j.ymssp.2023.109995

[9]. H. Bourekouche and H. Hadjer, "TinyML for Low-Power Embedded Intelligent Systems: A Review," SSRN, 2023. [Online]. Available: http://dx.doi.org/10.2139/ssrn.5431335

[10]. O. Vorgul and I. Svyd, "Neuron Networks Design in STM32 Cube," Microprocessors and Microsystems, pp. 56-59, 2023, doi:10.35598/mcfpga.2023.020.

[11]. M. Ç. Küçükalp, "Implementation of real-time digital signal processing algorithms using STM32F7xx series microcontrollers," M.S. thesis, Technical University of Liberec, 2024.

[12]. H. K. Mewada and B. Deepanraj, "Low-Power Embedded ECG Acquisition System for Real-Time Monitoring and Analysis," 2024 IEEE World AI IoT Congress, pp. 179-184, 2024, doi: 10.1109/AIIoT61789.2024.10578949.

[13]. S. Chatterjee et al., "ADC-Net: Attention-based Dense Convolutional Network for Underwater Image Enhancement," in Proc. 2023 5th Intl. Conf. on Advances in Computing, Communication Control and Networking, pp. 790-795, 2023, doi: 10.1109/ICAC3N60023.2023.10541728.

[14]. C. Meurer et al., "2D Estimation of Velocity Relative to Water and Tidal Currents Based on Differential Pressure for AUVs," IEEE RAL, 2020.

[15]. G. Antonelli et al., "Tracking control for underwater vehicle-manipulator systems with velocity estimation," IEEE J. Oceanic Eng., 2000.

[16]. S. Song et al., "Efficient Velocity Estimation and Location Prediction in Underwater Acoustic Sensor Networks," IEEE IoT Journal, 2022.

[17]. P. Batista, C. Silvestre, P. Oliveira and B. Cardeira, "Low-cost Attitude and Heading Reference System: Filter design and experimental evaluation," 2010 IEEE International Conference on Robotics and Automation, Anchorage, AK, USA, 2010, pp. 2624-2629, doi: 10.1109/ROBOT.2010.5509537.

[18]. Aparna Harindranath, Manish Arora, "Effect of sensor noise characteristics and calibration errors on the choice of IMU-sensor fusion algorithms," Sensors and Actuators A: Physical, Volume 379, 2024, 115850, ISSN 0924-4247. doi: 10.1016/j.sna.2024.115850.

[19]. P. Wu et al., "Real-time estimation of underwater sound speed profiles with a data fusion convolutional neural network model," Sensors, vol. 24, 2024, doi:10.3390/s24010000

[20]. H. Huang et al., "STNet: Prediction of Underwater Sound Speed Profiles with an Advanced Semi-Transformer Neural Network," Sensors, 2025. https://doi.org/10.3390/jmse13071370

[21]. J. Bao, X. Mu, Y. Xue, Z. Zhu, and H. Qin, "Robust multi-sensor fusion for unmanned underwater vehicle navigation," Measurement, vol. 217, p. 117450, 2025, doi:10.1016/j.measurement.2025.117450

[22]. P. Wu, W. Huang, Y. Shi, and H. Zhang, "An Attention-Assisted Multi-Modal Data Fusion Model for Real-Time Estimation of Underwater Sound Velocity," Remote Sensing, 2025. https://doi.org/10.48550/arXiv.2502.12817

[23]. R. Pike et al., "A Practical Guide for Noise Characterization of Back Pressure Sensors: Toward Digital Twin for an Industrial High-Horse Power Engine," Sensors, 2024. 10.1109/lsens.2024.3400009

[24]. V. S. Allu, "Implementation of Machine Learning Models on STM32 Microcontrollers," 2023. DOI:10.13140/RG.2.2.23493.26089

[25]. A. Khan et al., "A Comparative Analysis of LSTM Models with Attention and Squeeze and Excitation Blocks for Activity Recognition," IEEE Sensors Journal, 2025. https://doi.org/10.1038/s41598-025-88378-6