# *Morpheus*: An A-sized AUV with morphing fins and algorithms for agile maneuvering

Supun Randeni[1,*], Michael Sacarny[1], Michael Benjamin[1] and Michael Triantafyllou[1]

***Abstract*— We designed and constructed an *A-sized* base autonomous underwater vehicle (AUV), augmented with a stack of modular and extendable hardware and software, including autonomy, navigation, control and high fidelity simulation capabilities (*A-size* stands for the standard sonobuoy form factor, with a maximum diameter of 124 mm). Subsequently, we extended this base vehicle with a novel tuna-inspired morphing fin payload module (referred to as the *Morpheus* AUV), to achieve good directional stability and exceptional maneuverability; properties that are highly desirable for rigid hull AUVs, but are presently difficult to achieve because they impose contradictory requirements. The morphing fin payload allows the base AUV to dynamically change its stability-maneuverability qualities by using morphing fins, which can be deployed, deflected and retracted, as needed. The base vehicle and *Morpheus* AUV were both extensively field tested in-water in the Charles river, Massachusetts, USA; by conducting hundreds of hours of operations over a period of two years. The maneuvering capability of the *Morpheus* AUV was evaluated with and without the use of morphing fins to quantify the performance improvement. The *Morpheus* AUV was able to showcase an exceptional turning rate of around 25-35 deg $s^{-1}$. A maximum turn rate improvement of around 35% - 50% was gained through the use of morphing fins.**

## I. Introduction

The directional stability of autonomous underwater vehicles (AUVs) ensure the ability to maintain a steady course with minimal corrective control actions in the presence of disturbances [1]–[3]. The agility, or the maneuverability, of an AUV is the potential to make rapid maneuvers in heading and depth planes. The stability and agility have contradictory requirements; i.e. static or controlled surfaces located towards the stern of the vehicle (e.g. rudders, elevators, fixed fins, shrouds, etc.) increase the directional stability; however, they also adversely affect the maneuverability, reducing the ability to make rapid turns. This is because the increment in the stability index of a vehicle due to stern control surfaces is often larger than the turning moment it provides [4]–[11].

Both stability and maneuverability are desired features for AUVs [12]; therefore, AUVs in general are designed for a middle ground performance, partially compromising both stability and maneuverability. Improving both these features simultaneously was not possible for torpedo-shaped AUVs because they impose contradictory requirements. However, in our recent work [9], [11], we theoretically as well as experimentally showed that both stability and maneuverability can be improved by dynamically altering the directional stability, adopting the concept of bio-inspired morphing fins.

[1]Department of Mechanical Engineering, Massachusetts Institute of Technology, Cambridge, MA, USA

[*]supun@mit.edu

This work was supported by Lockheed Martin Corporation

### A. Designing an A-sized "base" AUV

An AUV is a complex system with a number of co-related subsystems. These subsystems can be primarily divided into two categories: (1) the base layer, and (2) the specialized layer. The base layer consists of components that are essential for basic autonomous operations of the vehicle (i.e. the base vehicle); for example, underwater navigation, basic autonomy, low-level control, basic communication, and related essential sensing capabilities. In general, AUVs are employed to conduct specific task(s) and mission(s). The specialized layer includes additional hardware and software components that are vehicle and application specific, which are built on top of the base layer. This layer may include additional hardware interfaces and drivers, sensor processing algorithms, autonomy algorithms, etc. For instance, a vehicle designed to conduct side-scan sonar mapping operations, the specialized layer will include the sonar related hardware components and specialized software modules such as sensor drivers, on-the-fly data processing and recording software, and potential autonomy algorithms for adaptive sampling and mapping.

In this work, we designed and constructed an *A-sized* base AUV, augmented with a stack of modular hardware and software, including navigation, autonomy, control and high fidelity software-in-the-loop (SITL) and hardware-in-the-loop (HITL) simulation capabilities (note – *A-size* stands for the standard sonobuoy [13] form factor, with a maximum diameter of 124 mm and a length of around 0.9 m, ensuring the ability to launch from standard sonobuoy launchers onboard a wide array of fixed wing and rotary wing air crafts, surface ships and submarines [14]). Subsequently, leveraged from our previous work [11], we extended the base vehicle with a new tuna-inspired morphing fin payload module design, where the fins that can be deployed, deflected and retracted as required, augmenting the vehicle with capability to dynamically change its stability-maneuverability qualities. The designed base vehicle with morphing fin payload mechanism is named as the *Morpheus* AUV.

### B. Designing a morphing fin payload module

Aquatic animals that specialize in cruising, such as tunas, require a higher directional stability to minimize the control action needed during cruising. Hence, they have streamlined bodies that are relatively stiff, limiting their body flexing to the last 30% of their length [15]. However, their prey consists

of smaller fish that have high body flexibility, and, hence, by employing significant body curvature, they can turn very rapidly [16]. As a result, tunas are capable of systematically changing the shape of their body fins to dynamically change the directional stability to conduct rapid maneuvers at high speed [17], [18] – when the forward located fins are retracted, their body becomes more directionally stable, gaining the ability to stably cruise at high speeds using a small amount of control energy. When they need to make a rapid turn, especially at high speeds, they deploy the dorsal fins whose mere presence destabilizes the body, increasing maneuverability. In addition, active control of the ventral fins provides additional turning moment, and smooth transients of forces and moments to obtain the precise level of directional body stability for the intended maneuver [19].

Inspired by tuna's adaptation mechanism, our previous work [9], [11] demonstrated the ability to implement an engineered design of retractable fins for rigid-hull, torpedo-shaped AUVs in order to dynamically vary the stability-maneuverability indices. While there is a number of recent studies and designs on developing biomimetic AUVs [20]–[23], as suggested by [24] and [25], there is still a large gap between fish-like vehicle platforms and their corresponding aquatic animal; hence, torpedo-shaped AUVs are still superior to biomimetic AUVs in terms of speed and endurance. In addition, many missions tasked to AUVs, such as seabed mapping, anti-submarine warfare, and surveillance, cannot be performed by biomimetic vehicles due to their unsteady large lateral motions, mechanical noise, and difficulty to design a sufficiently large payload bay while fulfilling soft-body bio-mimetic requirements. However, bio-mimetic AUVs are superior in terms of their maneuverability and agility, as compared to traditional torpedo-shaped vehicles [24]. Therefore, the intention of our work is not to develop a biomimetic AUV, but rather to create a bio-inspired vehicle by replicating the resultant hydrodynamic effects for a rigid-body engineered design, in order to enhance the vehicle's operational performance.

Through theoretical derivations, towing tank experiments and analytical simulations, [9] investigated the ability to alter the stability and maneuvering qualities of self-propelled, rigid hull AUVs by employing morphing fins. Our previous work [11] further extended this by investigating the variation of stability-maneuverability with different vehicle configuration and appendage designs. The evaluated vehicle configurations included: (1) the bare hull vehicle, (2) bare hull with different sizes of stern control surfaces, (3) different sizes of stern control surfaces combined with forward fins (4) different sizes of forward fins, and (5) different locations of the forward fins. This investigation was carried out by employing mathematical analysis, captive model tests and maneuvering simulations; validated with free swimming experiments. A 1-meter long bare hull AUV, retrofitted with different 3D-printed static appendages was used to investigate the variation of turning rate with free-swimming experiments.

In this paper, we present the design and construction of an A-sized base vehicle, including: (1) hull form and appendage configuration; (2) base vehicle hardware design, including the mechanical design and the construction of actuators, internal electronics and embedded computer system; (3) base vehicle software design, including underwater navigation, basic autonomy, low-level control, basic communication, and related essential sensing capabilities. Subsequently, leveraged from our previous work [11] in terms of hydrodynamic design, we develop an operational morphing fin payload module, and outfit it into our A-sized base vehicle; creating the *Morpheus* AUV. We present the design and construction of the morphing fin payload, including: (1) theoretical aspects; (2) mechanical and hardware design; and (3) software algorithms for adaptive control of the morphing fins. We demonstrate both the base vehicle and *Morpheus* AUV in-water, in the Charles river, Massachusetts, USA by conducting hundreds of hours of operations over a period of two years, evaluating the maneuvering capability of the vehicle with and without the use of morphing fins to quantify the performance improvement.

## II. Base Vehicle Hardware Design

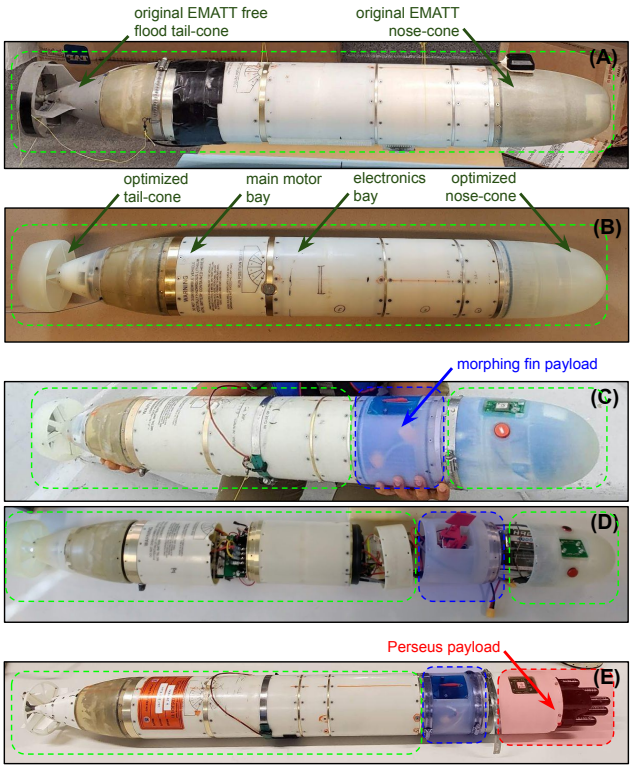


Fig. 1. The designed A-sized base AUV in four different configurations. Sections outlined in green, blue and red indicate base vehicle, morpheus payload and piUSBL payload components respectively. (A) The base vehicle in the conventional EMATT hullform; (B) base vehicle in the optimized hullform; (C) base vehicle appended with morphing fin payload (i.e. *Morpheus* AUV); (D) *Morpheus* AUV with disassembled hull sections; and (D) base vehicle appended with both morphing fin payload and piUSBL payload (i.e. *Perseus* AUV)

The base vehicle developed in this work was a derivation of the expendable mobile anti-submarine warfare training

target (EMATT) vehicle hullform, designed and produced by Lockheed Martin Corporation [14]. Figure 1A shows the first iteration of our base vehicle, which utilized an EMATT shell, augmented with our own electronics and software stacks. The shell included the original EMATT nose-cone, empty body shell, main-motor bay that housed the original EMATT main motor, and a free-flood tail-cone with solenoid controlled control surfaces. Throughout this paper, we refer to this vehicle as the "*MIT-EMATT*".

The second iteration of our base-vehicle; i.e the optimized edition, is shown in Figure 1B, which included hydrodynamically optimized nose and tail cones. The optimized nose-cone included an embedded GPS antenna, LED strobes, external pressure sensor and vacuum port; and the optimized tail-cone included four individually controlled, servo-driven control surfaces. These are further discussed in the following sections.

Figures 1C and 1D show the base vehicle appended with the morphing fin payload module. When our base vehicle is appended with the morphing fin payload it is referred to as the *Morpheus* AUV. The *Morpheus* AUV had an overall length of 0.9 m and an *A-sized* maximum diameter of 0.123 m. Figure 1E shows the base vehicle appended with morphing fin payload module as well as a piUSBL payload module [26], [27], which is referred to as the *Perseus* AUV.

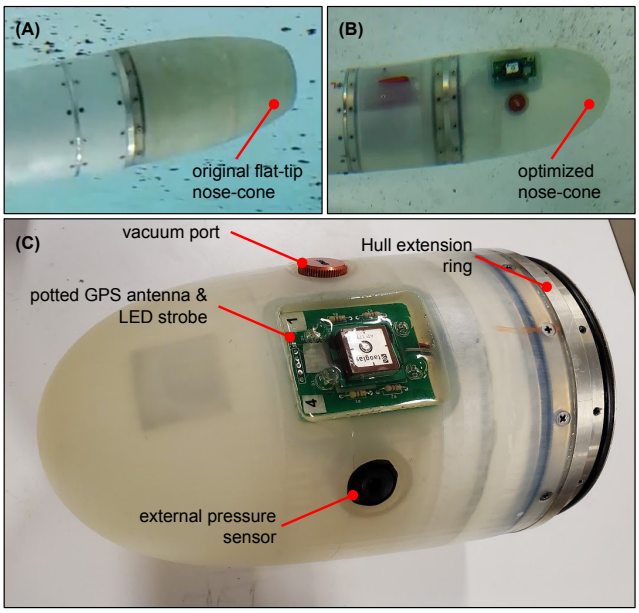


Fig. 2. Comparison between the (A) flat-tip nose-cone of the original EMATT platform, and (B) hydrodynamically optimized nose-cone. (C) The 3D printed optimized nose-cone consisted of two cylindrical slots to install the external pressure sensor and vacuum port; and a third rectangular slot to insert a circuit board containing the GPS antenna and LED strobes, which was potted with clear epoxy to make it waterproof.

### A. Nose-cone design

The original EMATT platform had a nose-cone with a flat-tip, primarily to ensure a higher usable space density, which resulted in a higher drag coefficient (see Figure 2A). In this work, a new nose-cone was designed as shown in Figures 2B and 2C, which was optimized to reduce the hydrodynamic resistance of the body as well as to preserve the usable space density inside the nose-cone [28].

The optimized nose-cone was manufactured by 3D printing; which was connected to the vehicle body using a metal hull extending ring, as shown in Figure 2C. The nose-cone was designed with two cylindrical slots, which were used to install the vehicle's external pressure sensor sensor (i.e. Blue Robotics depth sensor [29] housing an MS5837-30BA pressure sensor [30]) to measure the vehicle depth, and a vacuum port. A third rectangular slot was also designed, which allowed to insert a circuit board containing the GPS antenna and LED strobe. Upon installation of the circuit board, the slot was potted with clear epoxy to ensure the water-tightness. The vehicle's main battery bank was housed inside the nose-cone.

### B. Tail-cone design

The original EMATT platform had a tail-cone with a solenoid-controlled single rudder and a single elevator. Due to solenoid control, they both were limited to three positions: hard-to-port, hard-to-starboard and neutral. In the *MIT-EMATT* base-vehicle, we maintained the same actuator mechanism, connected to our own electronics and software.

The hydrodynamically optimized tail-cone version, as shown in Figure 3 had four cruciform-shaped, independently controlled, servo-driven control surfaces that can provide heading, pitch and roll control to the vehicle. Due to servo control, each control surface had the ability to be precisely controlled with a maximum articulation angle limit of around 15 degrees. The control surfaces and the propeller were protected by a shroud, which was an operational requirement. The shroud was connected to the hull of the vehicle using four fixed fins, which were not only acting as supports for the shroud, but their fixed 3° deflection angle also developed lift forces and hence a moment that counteracts the propeller torque.

As shown in Figure 3A, the tail-cone assembly consisted of the four control surfaces, their control linkages, and four servos, all mounted in a 3D-printed shell. The servos were held in place in the shell by 3D-printed dogs. Servo shaft rotation was converted to push-rod reciprocating action by cam assemblies. The push-rods then drove control surface articulation through control arms, while the control surfaces rotate around fixed pins. Since the tail-cone module was free-to-flood and the utilized micro-servos were not intended for submersion, they were oil-filled in-house. The servo cables were transitioned to the watertight main-motor bay through marine epoxy-filled bulkhead penetrators.

### C. Electronics design

Figure 4 represents the overall electronics design. Off-the-shelf electronic components, including a BeagleBone Blue single-board computer and motor controller were physically and electrically joined on a central custom backboard printed circuit board (PCB). Wiring then connected the backboard to

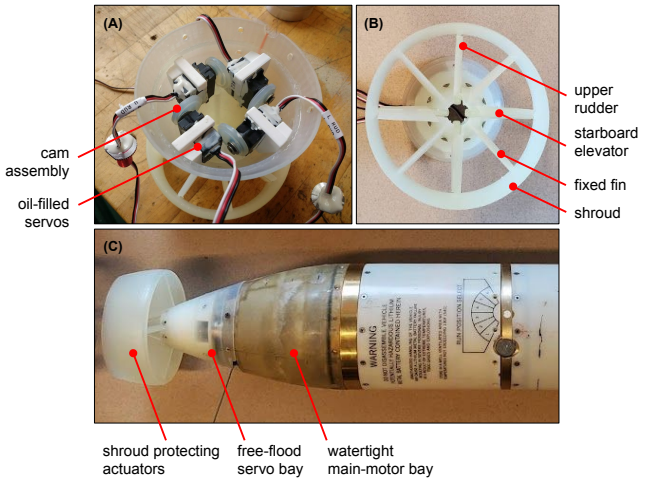


Fig. 3. The hydrodynamically optimized tail-cone with four independently controlled, servo-driven control surfaces. (A) The tail-cone assembly consisted of the four oil-filled micro-servos, cam and push-rod assemblies and control linkages, all mounted in a 3D-printed shell. (B) The two rudders, two elevators and propeller were protected by a shroud. The shroud was connected to the hull of the vehicle using four fixed fins, which had a fixed 3° deflection angle to counteract the propeller torque. (C) The free-flood, 3D-printed tail-cone section was separated from the watertight main-motor bay using a bulkhead, which had a shaft seal to penetrate the propeller drive shaft.

batteries, power controls, sensors, servos, and a custom LED strobe panel.

*1) BeagleBone Blue:*

The BeagleBone Blue single board computer [31] was selected as the main vehicle computer. It contains an ARM processor [32] that runs Linux operating system, as well as two Programmable Real-Time Units (PRUs), which are independent of the ARM processor; therefore, capable of quickly responding to inputs and produce very precisely timed outputs, such as PWM motor control outputs, similar to a micro-controller. Thus, both low-level control as well as higher-level processes were able to be run on a single computer, reducing the complexity and saving physical space that is precious for A-sized AUVs. The BeagleBone Blue included analog to digital converters (ADC), general purpose input and output (GPIO), PWM support, embedded inertial measurement unit (IMU), $I^2C$ interface, embedded WiFi, and universal asynchronous receiver-transmitter (UART) serial buses.

*2) Backboard:*

A custom PCB served as an electronic and physical integration foundation for the BeagleBone and other components, including the motor controller, current sensor, GPS module, 6V power supply, power conductors, servo connections, and system cabling. Discrete components provided ADC conditioning and logic translation for the LED strobe.

*3) Motor controller:*

The utilized main motor controller was a Cytron MD25HV 7-58V controller, selected to provide power in excess of 1 kW, should this be required for future designs, all in a compact and durable package. The standard EMATT motor required 150 W.

*4) Current sensor:*

A Pololu 4046 current sensor produced an analog signal that was used to monitor motor current.

*5) Depth and barometric pressure sensors:*

A Blue Robotics Bar30 depth sensor provided real-time depth readings via the $I^2C$ bus to the BeagleBone. The BeagleBone itself has an ambient pressure sensor, useful for pre- and mid-test internal vacuum monitoring.

*6) LED strobe:*

The LED strobe panel was located in the nose-cone. It consisted of LEDs and a GPS antenna mounted on a custom PCB, all potted in clear epoxy.

*7) Servo drive:*

The six micro-servos in the *Morpheus* vehicle design were driven by conventional PWM signaling from the BeagleBone through the backboard.

*8) GPS and cellular modem:*

GPS and cellular modem capabilities were provided by an Adafruit FONA 3G Cellular Breakout board. The antenna for this was routed to the nose-cone.

*9) Power supply:*

The nominal battery voltage for the system was 44.4 volts, supplied by two 6 cell lithium-polymer (LiPo) batteries in series. These were connected to the rest of the system by a 30A combination circuit breaker/power switch, followed by a relay-operated DC contactor. The contactor was only activated when an external plug is inserted into a through-hull connector. This allowed us to control vehicle power after the hull has been closed and pressurized.

Raw battery voltage was brought down to 12V by an automotive-style buck converter and distributed to the Bea-

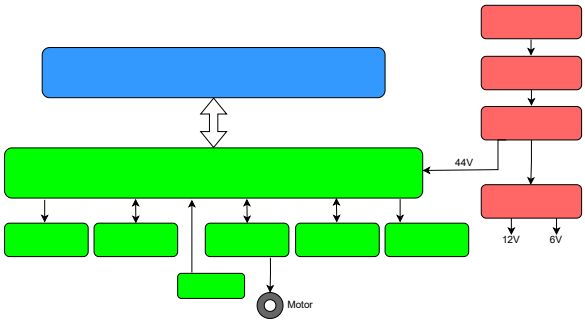


Fig. 4. A high-level overview of the electronics of the *Morpheus* AUV.

## III. Base Vehicle Software Design

In this work, we subdivided the critical components (both software and hardware) of the *Morpheus* AUV into two categories; the base vehicle components, and specialized

components. This modular subdivision allowed us to rationally reconfigure and re-purpose our A-sized base vehicles for other applications.

Figure 5 illustrates a higher level overview of hardware and software components of the *Morpheus* AUV, together with the information flow among them. The components that belong to the base layer are filled in blue, while those belong to the specialized layer (i.e. related to morphing fins, in this case) are filled in green. The remainder of this section will discuss the vehicle software components in the base layer.

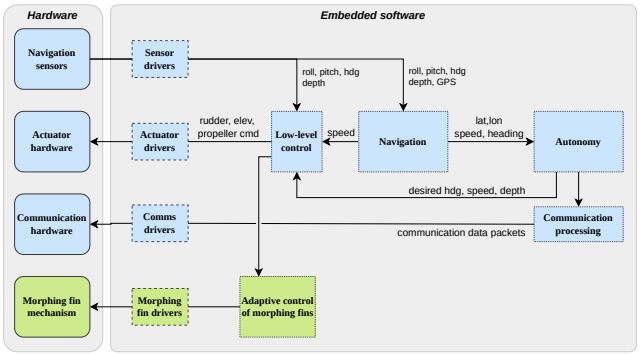


Fig. 5. A high-level overview of the hardware and software components of the *Morpheus* AUV, together with the data flow among them. The components that belong to the base layer are filled in blue, while those belong to the specialized layer (i.e. the components related to morphing fins) are filled in green.

The base software layer, shown in Figure 5 with blue filled blocks, is responsible for all the essential functionalities of a vehicle that transforms the hardware package into an autonomous platform. This includes navigation, autonomy, low-level control, communication and interfacing with related sensor and actuator hardware.

As seen from Figure 5, the sensor-to-actuator data flow begins with navigation sensor drivers, which communicate with external navigation sensor hardware, and provide raw sensor data to the vehicle's navigation system. The navigation software is responsible for estimating the vehicle's position (i.e. the latitude, longitude of the vehicle where it thinks it is at), depth, speed and attitude (roll, pitch and heading angles), while on the surface as well as underwater. The autonomy software is responsible for guiding the vehicle to the intended target location while avoiding no-go zones and obstacles in order to accomplish the mission tasks. The autonomous helm typically digests the navigation solution, and continually produces decisions on the desired heading, desired speed and desired depth commands that the vehicle needs to follow in order to achieve mission objectives. The low-level control software is responsible for executing the desired heading, speed and depth commands instructed by the autonomy system. This is achieved by controlling the actuators of the vehicle (e.g. propeller speed and control surface angle of attacks) to maintain the desired commands to the best of its ability. The actuator drivers then communicate these commands to the external actuator hardware; for example, using PWM, GPIO, and controller area network (CAN) bus commands.

With respect to our base vehicle software design, the components shown in Figure 5 are subdivided into three modular sub-systems. The primary software hub, which is referred to as the *MITFrontseat*, includes low-level sensor and actuator drivers, low-level control system, higher-level mission and safety management processes. The *MITFrontseat* outsources the navigation task to a specialized navigation engine – named *HydroMAN 2.0*, which provides the vehicle's position estimate (i.e. navigation solution) to *MITFrontseat*. The vehicle autonomy is either handled within *MITFrontseat*, or can also be outsourced to a user's own payload autonomy software system. Each sub-component outlined in this paragraph are further explained in the following sections.

### *A. Middlewares – inter-process and inter-system communication*

As visualized in Figure 5, the base vehicle software is composed of a number of distributed components; e.g. low-level interfaces, navigation, autonomy and low-level control modules. Hence, using a suitable middleware, or a combination of several middlewares, to glue the software components together is important [33], [34]. There are a few different choices of middleware typically used by the marine robotics community, such as, common object request broker architecture (CORBA) [35], mission oriented operating suite (MOOS) [36], data distribution service (DDS) [37], robotics operating system (ROS) [38], Goby3 [39], lightweight communications and marshalling (LCM) [40], etc. In this work, we primarily use MOOS as the middleware for inter-process communication within the software sub-systems. In addition, a standardized interface that exchanges pre-defined, encoded google protocol buffer (protobuf) [41] messages over the TCP communication architecture is also used for inter-system communications; for example, between *MITFrontseat* and *HydroMAN 2.0*, and between *MITFrontseat* and payloads. This architecture was chosen to ensure the independence of each sub-system from others, and the ability to re-use these sub-systems in different frameworks, even if they use non-MOOS middleware. This is further discussed in Sections III-D.1 and III-I.

### *B. Embedded computing system*

Figure 6 illustrates the complete base-vehicle software diagram of the *Morpheus* AUV. In this work, we developed a boilerplate frontseat software stack, referred to as the *MITFrontseat*, that is sufficiently modular and generic to be utilized as a base-vehicle frontseat software for other types of AUV designs as well. As seen from Figure 6, the *MITFrontseat* stack handles both low-level routines such driving low-level hardware (i.e. driving and communicating with sensors and actuators), while also handling higher-level processes such as navigation, autonomy, control and vehicle safety management. Typically, a micro-controller is used to handle low-level routines, which is interfaced with a single board computer that runs higher level processes. In this work, however, we have used a BeagleBoard single board computer

[31] as the main vehicle computer. It contains an ARM processor [32] that runs Linux operating system, as well as two PRUs, which are independent of the ARM processor; therefore, are capable of quickly responding to inputs and produce very precisely timed outputs, such as PWM motor control outputs, similar to a micro-controller. Thus, both low-level routines as well as higher-level processes were able to be run on a single computer, reducing the complexity and saving physical space that is precious for A-sized AUVs.

Since the main vehicle computer also handles low-level hardware, the *M*ITFrontseat included an array of MOOS drivers for various sensors and actuators that are typically used in micro AUVs. One of the drawbacks of this architecture is that some of these MOOS drivers are only supported for BeagleBoard computers; hence, if one anticipates to run *M*ITFrontseat on a different single board computer board, these low-level drivers may required to be modified accordingly. That said, all the higher level processes (i.e. navigation, autonomy, control and vehicle and missions safety algorithms) are agnostic to the computer board.

### C. Sensor software drivers within MITFrontseat

As seen from Figure 6, an array of MOOS drivers were developed to communicate with various hardware sensors and actuators used in the vehicle via various hardware interfaces available onboard the BeagleBoard. Each sensor driver publishes the raw sensor data to the MOOS database of the *M*ITFrontseat MOOS community.

*1) Depth sensor driver (*`iBlueRoboticsDepth`*):*

The depth of the vehicle was obtained by measuring the external hydrostatic pressure, and converting it to a corresponding depth value, accounting for the water temperature and density. The *Morpheus* vehicle was equipped with a Blue Robotics depth sensor [29] that housed an MS5837-30BA pressure sensor [30]. The pressure sensor was connected to the $I^2C$ bus of the BeagleBone Blue embedded computer using its JST connectors.

A MOOS driver application, `iBlueRoboticsDepth`, was developed to read the external pressure and temperature measurements from the Blue Robotics pressure sensor, and to compute the corresponding vehicle depth, using a pre-configured water density. The calculated depth was then filtered with an outlier rejection scheme – if a depth reading that suggests a depth rate of over 5 m $s^{-1}$ (pre-configurable) was observed, it is rejected since a 5 m $s^{-1}$ depth rate is unrealistic. Upon the outlier rejection scheme, a moving average filter with a window size of 20 samples (pre-configurable), which results in a time interval of around 2 seconds, is also applied to smoothen the data. Filtered depth value is finally published to the MOOS database.

*2) IMU sensor driver (*`iBBBlue` *and* `iXsensMTi`*):*

The attitude (i.e. the roll, pitch and heading angles) of the vehicle was measured using an InvenSense MPU-9250 [42] micro-electromechanical system (MEMS) 9-axis inertial measurement unit (IMU), embedded in the BeagleBone Blue computer board, which is routed to the $I^2C$ bus.

An existing, thirdparty MOOS driver application, `iBBBlue` [43] was used to read data from the IMU. The `iBBBlue` application utilizes several functions given in the Robot Control Library [44] to conduct tasks such as reading IMU data from the $I^2C$ bus, IMU calibration correction and fusion of acceleration, angular velocity and magnetic intensity data to compute the roll, pitch and heading of the sensor. These raw data are then published to the MOOS database.

For applications that require more accurate attitude and heading information as compared to the InvenSense MPU-9250 [42] sensor, an external IMU sensor can be utilized, together with a MOOS driver for the sensor. An example is the Xsens MTi-3 [45] IMU unit, which can be connected to the BeagleBone using the UART interface. A MOOS driver, `iXsensMTi`, was created to read fused attitude data from the sensor, and publish them to the MOOS database.

*3) GPS and cellular modem driver (*`iAdafruitFona`*):*

An Adafruit FONA cellular breakout board [46], which contains a SIM5320 cellular module with an integrated GPS receiver [47] was used as the GPS and cellular modem of the vehicle. This module is connected to the vehicle's BeagleBone Blue computer using the UART interface. A MOOS driver application, `iAdafruitFona`, was developed to publish raw GPS data to the *MITFrontseat* MOOS database. This application also acts as a service that sends and receives short message service (SMS) text messages. Any incoming SMS messages from allowed phone numbers (pre-configured) are published to the MOOS database with the message content and sender's phone number. Any application within the *MITFrontseat* community can publish a specific MOOS variable to the database, containing the message content and phone number to forward the message to an outside phone number. For example, this service can be used to send an SMS to the vehicle operator's phone number with the GPS coordinates, upon mission completion and surfacing.

While the SIM5320 module also has the internet tethering capability, which could enable remote login to the BeagleBone Blue computer via the cellular network, this was not implemented in this work.

*4) Battery and power management system driver (*`iBBBlue`*):*

A custom battery and power management system was developed and integrated to monitor the main motor current draw and main battery voltage. The current and voltage values are provided to the BeagleBone Blue via its analog-to-digital converters (ADC). The `iBBBlue` reads these values are publishes them to the *MITFrontseat* MOOS community, which is used by `pFrontseatManager` for vehicle safety management.

*5) Monitoring the internal pressure:*

Typically, the air inside the vehicle pressure hull is partially pumped out through a vacuum port (see Section II-A) upon vehicle assembly. Once the air is pumped out, the vacuum port is closed, leaving a low pressure zone inside

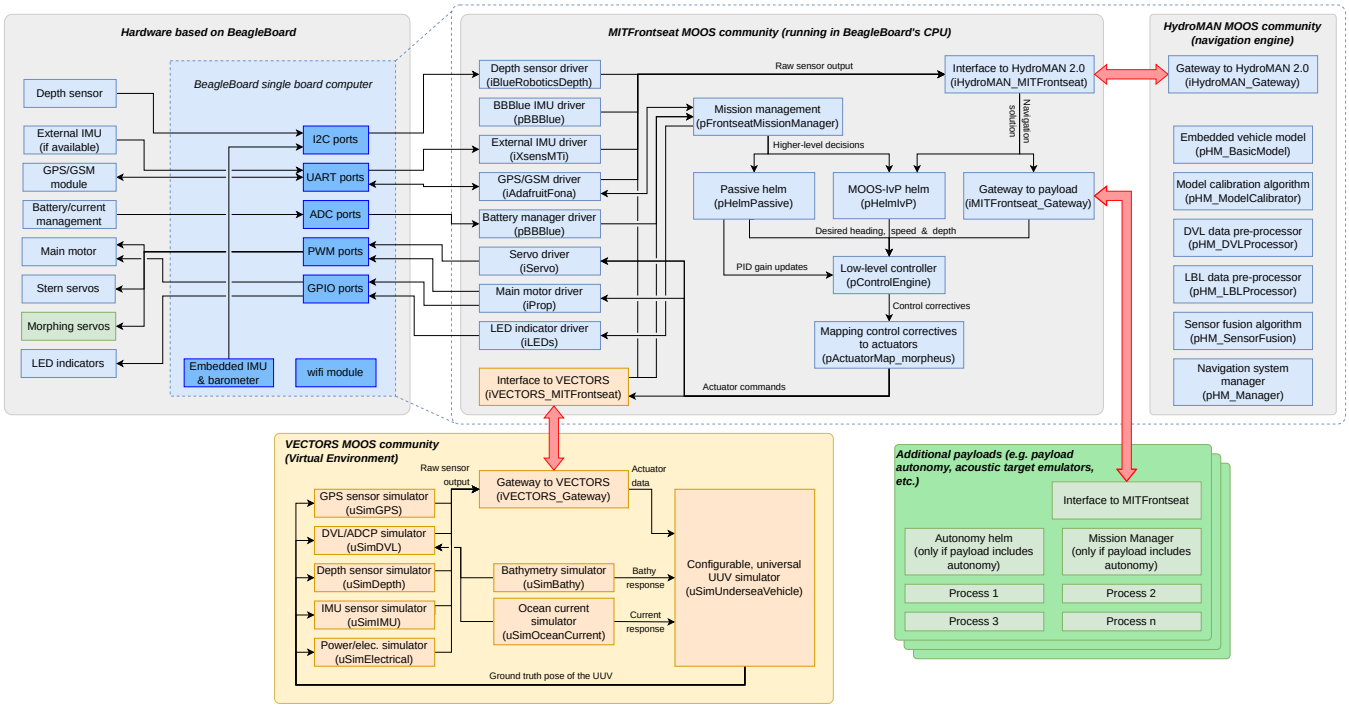


Fig. 6. A complete

the hull. If the hull is watertight, the internal pressure will be holding. A raise in the internal pressure indicates a leak in the pressure hull.

In the base vehicle, the air pressure inside the pressure hull is measured using the barometer embedded in the BeagleBone Blue computer. The barometer reading can be monitored upon vehicle assembly using functions and scripts given in the Robot Control Library [44].

### *D. Navigation software - HydroMAN 2.0*

In our software architecture, the *MITFrontseat* outsources the navigation task to an independent navigation engine, similar to most commercial AUVs (i.e. most AUVs rely on commercially available INS units for navigation – an INS is a combination of an IMU and a computer running a 'black-box' navigation fusion algorithm that fuses the IMU measurements with external sensors such as GPS, DVL, depth, USBL, etc. [48]–[50]). In this work, we utilize *HydroMAN 2.0* as the navigation engine.

*HydroMAN* (stands for hydrodynamic model aided navigation) is a self-learning, independent underwater navigation engine [51]–[56]. The HydroMAN 2.0 synthesizes raw measurements from sensors such as IMU, DVL, CVL, LBL/USBL, terrain-aided navigation and GPS into its self-calibrating vehicle flight dynamic model to compute the navigation solution, with the use of an array of sensor pre-processors and a layered extended Kalman filter based fusion algorithm. When accurate sensor measurements are available; for example, DVL bottom-lock and/or acoustic position updates, the HydroMAN self-calibrates the vehicle model to the local operating environment, largely compensating for the navigation drift provided by underwater currents and the flight dynamic model's own error estimate. The calibrated vehicle model is then utilized for navigation aiding when accurate sensors are unavailable, or turned off in order to save power.

Expensive tactical and navigation grade INS units and DVLs are infeasible for low-cost micro-AUVs that are limited by the cost, such as *Morpheus*. Therefore, they typically rely on inexpensive MEMS IMUs and an RPM-to-speed table for dead-reckoning based navigation, resulting in poor navigation performance. For such vehicles, *HydroMAN* uses a pre-identified vehicle flight dynamic model to estimate the vehicle velocity in surge, sway and heave directions, by using a combination of input variables such as the propeller speed, control surface angles and the rates of changes of roll, pitch and heading angles. As shown in [56] and [55], the vehicle dynamic model is capable to improving the navigation accuracy by orders of magnitude as compared to traditional RPM-to-speed curve based dead-reckoning. In the meantime, *HydroMAN* also provides the possibility to extend the base vehicle with additional navigation sensors if one anticipates to, without requiring changes to the navigation software stack.

HydroMAN is comprised of a number of MOOS applications as described in this companion paper [51]. Following subsections briefly summarize key functionalities of these sub-component:

*1) The interface to the HydroMAN MOOS community (*`iHydroMAN_Gateway`*):*

The *HydroMAN* version 2.0 is an independent navigation engine that interfaces with client systems (i.e. *MITFrontseat* in this case) using a TCP network connection. Therefore,

the front-end of *HydroMAN 2.0* is similar to internal fusion engines of tactical and navigation grade INSs – the client system can send external raw sensor data to *HydroMAN*; and *HydroMAN* will provide the fused, model-aided navigation solution. Hence, the *HydroMAN* is independent of the client's software architecture.

`iHydroMAN_Gateway` application runs within the HydroMAN MOOS community, serving as the gateway to *HydroMAN*. All incoming messages to HydroMAN (e.g. raw sensor data) and outgoing messages from HydroMAN (e.g. final navigation solution) are handed over by `iHydroMAN_Gateway`. It runs a TCP server, which allows the client system's HydroMAN driver (in this case, `iHydroMAN_MITFrontseat` MOOS application that runs within the *MITFrontseat* MOOS community) to connect as a TCP client, and exchange messages using a google protocol buffer based standardized message definition. This standardized message definition and server-client architecture ensures the independence of HydroMAN; i.e., the client system's HydroMAN driver does not necessarily need to be a MOOS application.

*2) Vehicle flight dynamic model (*`pHM_BasicModel`*):*

As virtual navigation aiding sensor, an embedded AUV flight dynamic model, based on the principle of conservation of energy [55] is used in HydroMAN to estimate the linear velocities of the AUV (i.e. $u$, $v$ and $w$). While the actual structure of the vehicle dynamic model varies from vehicle to vehicle, Equations 1-3 demonstrate how the propeller speed ($RPM_{(t)}$), measured vehicle angular velocities ($p_{(t)}$, $q_{(t)}$ and $r_{(t)}$), and the linear velocities estimated at the previous timestamp ($u_{(t-1)}$, $v_{(t-1)}$ and $w_{(t-1)}$) are used to derive the vehicle velocities at the current timestamp (i.e. $u_{(t)}$, $v_{(t)}$ and $w_{(t)}$) for an example vehicle:

$$\begin{aligned} u_{(t)} = \alpha_1 RPM_{(t)} + \alpha_2 RPM^2_{(t)} + \alpha_3 q_{(t)} p^2 + \\ \alpha_4 r_{(t)} v^2_{(t-1)} + \alpha_5 q_{(t)} w^2_{(t-1)} + \alpha_6 p^2_{(t)} + \alpha_7 q^2_{(t)} + \\ \alpha_8 r^2_{(t)} + \alpha_9 p_{(t)} r^2_{(t)} + \alpha_{10} z_{(t)} \end{aligned} \quad (1)$$

$$\begin{aligned} v_{(t)} = \beta_1 q_{(t)} p^2_{(t)} + \beta_2 p^2_{(t)} + \beta_3 r_{(t)} u^2_{(t-1)} + \\ \beta_4 q_{(t)} u^2_{(t-1)} + \beta_5 q^2_{(t)} + \beta_6 r^2_{(t)} + \beta_7 p_{(t)} r^2_{(t)} + \beta_8 z_{(t)} \end{aligned} \quad (2)$$

$$\begin{aligned} w_{(t)} = \gamma_1 r_{(t)} u^2_{(t-1)} + \gamma_2 q_{(t)} u^2_{(t-1)} + \gamma_3 q_{(t)} p^2_{(t)} + \\ \gamma_4 p^2_{(t)} + \gamma_5 q^2_{(t)} + \gamma_6 r^2_{(t)} + \gamma_7 p_{(t)} r^2_{(t)} + \gamma_8 z_{(t)} \end{aligned} \quad (3)$$

where $\alpha_n$, $\beta_n$, and $\gamma_n$ are AUV-dependent flight dynamic model parameters that were estimated using a real-time recursive least squares system identification algorithm. The derivation of the dynamic model, model optimization and parameter estimation procedures are beyond the discussion of this paper, and further details are given in [55].

The velocity and position estimated by the flight dynamic model are relative to the water column (i.e. $\nu^{model}_{(\text{auv}|\text{water})}$ and $x^{model}_{(\text{auv}|\text{water})}$) since the model excludes water currents. Therefore, the error sources of the model-based velocity and position estimates include the drift due to water currents and the uncertainty of the model [55], [56], which are counteracted by the self-adaptation of the flight dynamic model

*3) Self-calibration of the flight dynamic model to the operating environment (*`pHM_ModelCalibrator`*):*

The uncertainty of the dynamic model, and water current velocity ($\nu_{(\text{water}|\text{earth})}$) are estimated on-the-fly within `pHM_ModelCalibrator` when accurate sensor measurements such as DVL bottom-lock and/or acoustic navigation updates are available. These estimates are used to convert the model-based velocity from water reference to earth reference, as detailed in Equation 4:

$$\nu^{adaptM}_{(\text{auv}|\text{earth})} = \nu^{model}_{(\text{auv}|\text{water})} + \nu^{adaptM}_{(\text{water}|\text{earth})} + \sigma^{adaptM}_{\nu^{\text{model}}} \quad (4)$$

Two self-calibration strategies are available within `pHM_ModelCalibrator`: (a) using acoustic position updates and (b) using the bias error of the model-based velocity (i.e. bias error estimated by the error-state extended Kalman filter (EKF) of the fusion algorithm). Further details on these algorithms are given in [51].

*4) DVL pre-processor (*`pHM_DVLProcessor`*):*

This MOOS application processes raw sensor measurements from velocity aiding sensors such as DVL and CVL. By considering the configured orientation of the sensor, the velocity measurements are transformed to the *HydroMAN* standard axis convention. An orientation mismatch detection mechanism is implemented to warn the operator and/or execute vehicle safety protocols if the configured sensor orientation is detected to be not accurate.

In under-ice operations, if the sensor is in an upward-facing configuration, measuring the velocity of the AUV relative to surface ice, `pHM_DVLProcessor` is capable of counteracting the velocity for potential drifts in the surface ice (i.e. surface ice in the Arctic is translated and rotated by wind and current forcing [57], and this ice drift velocity can be up to around 1 m $s^{-1}$, which can cause considerable navigation drift [58]). `pHM_DVLProcessor` can be aided with ice drift velocity information obtained from actual measurements (e.g. measured by a GPS unit located on the surface, and transmitted down to the vehicle via an acoustic link) or from modeling approaches [59].

*5) LBL pre-processor (*`pHM_LBLProcessor`*):*

Navigation aiding information provided in the form of position updates; for examples, acoustic position updates from LBL/USBL/SBL systems, terrain-aided navigation updates, etc. are pre-processed by the `pHM_LBLProcessor` application.

Some types of acoustic position updates (e.g. two-way-travel-time systems) can typically be outdated by more than 20 s when the position update is received by the AUV (i.e. $t - t_N$, where $t$ is the current timestamp). A 20-second time-lag could develop a position error of up to around 32 m (assuming a speed of 1.6 m $s^{-1}$); hence, are typically rejected by most commercial INS sensor fusion algorithms. `pHM_LBLProcessor` contains an algorithm to extrapolate

such position updates to the current timestamp using the self-adapting vehicle flight dynamic model as given in Equation 5:

$$x^{lbl}_{(\mathrm{t})} = x^{lbl}_{(\mathrm{t_N})} + \left( x^{adaptM}_{(\mathrm{t})} - x^{adaptM}_{(\mathrm{t_N})} \right) \tag{5}$$

where $x^{adaptM}_{(T)}$ is the vehicle position from the self-adapting flight dynamic model at timestamp $T$. The current timestamp is given by $t$ and the LBL timestamp is given by $t_N$.

This pre-processor allows *HydroMAN* to effectively utilize navigation updates that are time-lagged by large time periods (i.e. more than 5-minutes).

*6) Sensor fusion engine (*`pHM_SensorFusion`*):*

The sensor fusion application consists of two EKFs: (a) the error-state EKF that estimates the bias errors of the sensors, and (b) main-state EKF that fuses the bias error corrected measurements to obtain the final navigation solution.

The error-state EKF computes a running estimate of the bias errors of velocity sensors (e.g. DVL, CVL, etc.) and flight dynamic model in a layered pattern, in the hierarchy of the accuracy of the sensor. That is, the outlier removed acoustic position updates are first used to compute the bias error estimate of the DVL sensor, which is used to correct the DVL measurements. The bias error corrected DVL measurements are then used to compute the bias error of dynamic model, and other velocity aiding sensors, in a hierarchical order. Since the variation of bias error is generally a slowly changing function, this method allows *HydroMAN* to maintain a good navigation accuracy by using bias corrected dynamic model, even in events where the DVL drops out or turned off for a long period of time.

The main-state EKF included six states — the three dimensional velocity and position vectors. These states were estimated by fusing the bias corrected DVL, flight dynamic model and other velocity measurements together with the depth and position based navigation updates. More information regarding this layers sensor fusion approach is given in [51].

*7) Navigation manager (*`pHM_Manager`*):*

The *HydroMAN* system consists of a number of navigational safety management systems; e.g. EKF re-initialization when large navigation drifts are detected, executing safety protocols in situations where the filter is diverging due to faulty sensors, etc. The `pHM_Manager` application manages these features and publishes the final navigation solution.

### *E. Autonomy software*

Unlike unmanned sea-surface, ground, and aerial vehicles, AUVs cannot be remotely controlled due to the low bandwidth in acoustic communications; they must make decisions autonomously. Remote control, or teleoperation, in land, air, or surface vehicles may be viewed as a means to allow conservative, risk-averse operation with respect to the degree of autonomy afforded to the vehicle. In underwater vehicles, similar conservative tendencies are realized by scripting the vehicle missions to be as predictable as possible. Missions typical of early-model UUVs were composed of a preplanned set of waypoints accompanied by depth and perhaps speed parameters. The onboard sensors merely collected data that were analyzed after the vehicle was recovered from the water. However, improved sensor processing methods, embedded computing power, underwater navigation performance and adaptive and collaborative autonomy technology has enabled advanced autonomy for AUVs [60].

The base vehicle software stack that we developed carries several autonomy capabilities of several fidelity levels: (1) primitive missions with scripted decision outputs; (2) autonomous decision making with MOOS-IvP behavioural helm that runs on the *MITFrontseat* MOOS community; and (3) payload autonomy where the *MITFrontseat* ingests decision commands from thirdparty payload-based autonomy systems.

*1) Higher level autonomy management (*`pFrontseatManager`*):*

The autonomy helm of the vehicle, regardless of the fidelity level, sits beneath and bound by a safety envelope set by this mission management application. In addition to enforcing safety rules, this application is also responsible for executing and switching autonomy behaviors with the use of a state machine.

The frontseat mission manager enforces vehicle-dependant and cruise-dependant safety rules, set by the operator during pre-launch mission configuration. The vehicle-dependent rules ensure the integrity of structural and electrical components and water-tightness of the vehicle by administering variables such as the maximum vehicle diving depth, minimum operating battery voltage, maximum operating motor current, maximum internal pressure. When a specific rule is violated, the vehicle mode will be autonomously switched to an orchestrated safe mode, depending on the violated rule. The cruise, or mission dependent rules include: (a) mission start time – the main motor start time could be delayed by a pre-configured time period since the mission launch; (b) mission end time – the mission manager leases the vehicle's control authority to the autonomy helm only for a pre-configured temporary time period; beyond which, the mission ending mode is executed; (c) maximum cruise depth – if the maximum safe operating depth for the cruise region is below the maximum diving depth of the vehicle.

When MOOS-IvP helm is run within the *MITFrontseat* MOOS community, the frontseat mission manager functions as a mission commander that carries out on-board command and control of mission behaviors. That is, with the use of a state machine, this application controls which IvP helm behaviors are spawned at a given time [60] and how they are switched between. The switching of IvP behaviors is either conducted completely autonomously, or manually triggered via communication methods detailed in Section III-F.

*2) Passive helm (*`pHelmPassive`*):*

The passive helm allows scripting of a timetable of predefined helm decisions (i.e. desired speed, desired heading, desired depth and vehicle mode command); each against a corresponding execution time (i.e. mission legs). During

the mission, pHelmPassive reads the pre-configured helm decisions from the configuration block, and posts to the MOOS database. Therefore, this primitive helm can be run without a vehicle navigation solution, making it a useful tool for preliminary testing of the vehicle.

Another key use case of the passive helm is for tuning of the vehicle's low-level control system. Most AUVs still use proportional-integral-derivative control systems for their low-level control; and fine-tuning them, which is typically done trial-and-error, is rather dull process. During this process, the autonomy system is required to first command a constant heading, speed and depth; followed by a step-change command in the either heading, speed or depth, depending on which degree-of-freedom is being tuned. The passive helm is an ideal tool for such simple, pre-dictated missions. In addition, passive helm also allows the users to configure PID gain changes during legs, which is ingested by the control engine as runtime PID gain updates. This functionality allows the operators to test multiple PID gain settings during a single mission, expediting the time consuming tuning process by orders of magnitude. Listing 1 shows a sample mission configuration block of pHelmPassive, where the P-gain of the vehicle's heading controller is updated during the third leg.

```
1 ADD_LEG: start_time=120, heading=180,
      speed=1.5, depth=1.5
2 ADD_LEG: start_time=240, heading=250,
      speed=1.5, depth=2.0
3 ADD_LEG: start_time=410, heading=250,
      speed=1.5, depth=2.0, heading_kp=0.8
4 ADD_LEG: start_time=420, heading=180,
      speed=1.5, depth=2.0
5 ADD_LEG: start_time=600, heading=250,
      speed=1.5, depth=1.5
```

Listing 1. A sample mission configuration block of pHelmPassive. This primitive mission updates the proportional gain of the vehicle's heading PID controller during the third leg.

*3) MOOS-IvP autonomy helm (*pHelmIvP*):*

The MOOS-IvP helm runs as a single MOOS application and uses a behavior-based architecture for implementing autonomy. Behaviors are distinct software modules that can be described as self-contained mini-expert systems dedicated to a particular aspect of overall vehicle autonomy. The helm implementation and each behavior implementation expose an interface for configuration by the user for a particular set of missions. This configuration often contains particulars such as a certain set of waypoints, search area, and vehicle speed. It also contains a specification of mission modes that determine which behaviors are active under which situations and how states are transitioned. When multiple behaviors are active and competing for influence of the vehicle, the IvP solver is used to reconcile the behaviors. More information regarding MOOS-IvP can be found from [60] and [61].

The *Morpheus* base vehicle software stack allows the users to run MOOS-IvP autonomy from within the *MIT-Frontseat* MOOS community. In this architecture, required behaviors are loaded to the mission configuration block, and the pFrontseatManager application acts as the mission commander in-charge of spawning and switching between behaviors.

*4) Payload autonomy:*

The main idea in the payload autonomy paradigm, or the backseat driver is the separation between vehicle control and vehicle autonomy. The vehicle control system runs on a platform's main vehicle computer, and the autonomy system runs on a separate payload computer. This separation is also referred to as the mission controller – vehicle controller interface. A primary benefit is the decoupling of the platform autonomy system from the actual vehicle hardware [60], [62]–[67].

The payload autonomy capability is built-in to the *Morpheus* base vehicle software stack through the development of a standard payload interface. This interface allows to forward any information posted in the *MITFrontseat* MOOS community to a payload computer via a TCP connection, using a standard protobuf message scheme; and the payload autonomy system is able to send autonomy commands (e.g. desired heading, desired speed and desired depth commands) back to the *MITFrontseat* via the same interface. More information regarding the interface is given in Section III-I. Hardware-wise, the payload autonomy system could run either on the same main vehicle computer or on a separate autonomy computer. In the case of *Morpheus*, the payload autonomy system was also run on the same main vehicle computer (i.e on the BeagleBone Blue) in order to conserve space inside the vehicle.

In the payload autonomy mode, as discussed in Section III-E.1, the pFrontseatManager leases the command of the vehicle to the payload autonomy system for a pre-configured time period. However, the payload autonomy system is still bound by the safety envelope set by the frontseat manager. If one anticipates to take unconditional control of the vehicle, this can be done by turning off safety parameters within pFrontseatManager.

### *F. Communication software*

Communication of low-cost, micro AUVs such as *Morpheus* can be generally classified into three categories: (1) short-range surface communication; (2) long-range surface communication; and (3) underwater acoustic communication.

Short-range surface communication is generally via a wifi network connection with the topside network. *Morpheus* vehicle achieves this using the BeagleBone Blue computer's embedded wifi modem. The computer's network settings are configured such that it connects to a specific wifi network whenever the vehicle is in range. This network is typically used to access the vehicle computer in order to conduct operations such as system testing, launching missions, debugging, data transfer, etc.

In *Morpheus* vehicle, long-range surface communication is achieved via the cellular network; with the use of SMS messages with a dedicated topside cellular phone. Very basic command and control, and vehicle status monitoring can be achieved with this service (e.g. this service is typically

configured to send an SMS to the topside phone with the GPS coordinates, upon mission completion and surfacing). We expect to expand the long-range surface communication capability by establishing a remote connection between the topside computer and vehicle with the use of cellular internet tethering for more advanced surface command and control, and telemetry monitoring; though the use of Goby-Acomms library [68], [69] for marshalling and dynamic priority queuing; and Goby liaison as the command and control GUI [70].

At the time of writing, the *Morpheus* vehicle is not equipped with an acoustic modem that enables transmission of datagrams while underwater. However, the piUSBL system in the *Perseus* payload allows very basic underwater command and control by switching the transmitter between various broadcast linear frequency-modulated chirps; each corresponding to different pre-defined vehicle autonomy behaviors [27]. In the future, we plan to expand the electronics stack of the vehicle with a small acoustic modem (e.g. [71]–[73]) for more advanced underwater command and control, and telemetry; through the use of Goby-Acomms library, which will also be used for long-range surface communication.

### G. Low-level control software

The low-level control software is responsible for executing the decisions commanded by the autonomy system; such as, desired heading, desired speed, desired depth and desired glide angle (i.e. in the case for gliding vehicles) commands. The vehicle control system executes such commands by controlling the vehicle-specific actuators such as the propulsion thrusters, control surfaces, buoyancy engines and weight shifting mechanisms, etc. Hence, the low-level control system of an AUV is generally platform-dependant. In *MITFrontseat*, we have generalized the control system by sub-dividing it to three components as shown in Figure 7: (1) a platform-independent control engine that produces control correctives in roll, heading, speed, and pitch; (2) a platform-dependant actuator mapper application that maps the control engine outputs to the actuator configuration of a specific vehicle; and (3) actuator drivers that produces low-level signals such as PWM, GPIO and CAN bus messages that drive the actuators.

*1) Control engine (*`pControlEngine`*):*

The low-level control engine of *MITFrontseat* consists of a set of single loop (i.e for heading, speed and roll sub-systems) and multi-loop (i.e. for the depth sub-system) PID control blocks that produce control corrective outputs. Control corrective outputs are essentially in the same order as actuator commands; e.g. control surface angle commands. In order to ensure the platform-independence of the control engine, PID outputs are published as control correctives; and the mapping of control correctives to actuators of a specific vehicle is conducted in a separate MOOS application.

For some vehicle hardware designs, there could be an offset between the IMU mount axis and vehicle axis. In *MITFrontseat*, this offset correction is carried out in the control engine. The axis offset correction for heading is done by pointing the vehicle's nose towards north, and poking a given MOOS variable. Similarly, roll and pitch offsets are corrected by keeping the vehicle at zero roll and pitch, and poking two separate MOOS variables. These offsets are written to a configuration text file, which is read on start-up to correct the IMU offset.

The control engine contains three independent single-loop PID control blocks for heading, speed and roll sub-systems. They ingest the difference between the desired and actual values (i.e. for example, the heading error), and compute a PID corrective that would attempt to minimize the error, with the use of configured PID gains.

For under-actuated vehicles such as flying type AUVs, the depth DOF cannot be directly controlled; and is rather controlled by varying the pitch angle of the vehicle. Thus, a two-loop PID controller is implemented for the depth sub-system. As seen from Figure 7, the depth PID control block produces a depth control corrective, which becomes the desired pitch input for the pitch PID control block. The latter then computes the pitch control corrective, which is sent to the actuators that control the pitch DOF of the vehicle (e.g. elevators). However, for gliding vehicles, the optimized desired glide angle (i.e. desired pitch value) is provided by the autonomy system. For such vehicles, the control engine by-passes the depth PID block; and the pitch PID block uses the desired glide angle as the desired pitch value.

The *MITFrontseat* is compatible for vehicle with multiple modes; for example, for hybrid gliders that has both propelled and gliding modes; and for amphibious vehicles that are capable of operating in-water as well as ashore. For such multi-mode vehicles, multiple control settings are typically required; for instance, in the case of hybrid vehicles, one PID setting for the propelled mode (i.e. regulating the propeller and control surface angles to control the speed, pitch and heading), and another PID setting for the gliding mode (i.e. regulating the buoyancy engine and battery-pack position to control the same variables). As shown in Figure 7, the control engine handles this by dynamically creating an 'N' number PID controller sets during start up, each set corresponding to a vehicle mode. When the autonomy system switches to a specific vehicle mode, the control engine also switches itself to the corresponding PID controller and gain setting.

*2) Mapping control correctives to vehicle actuators (*`pActuatorMap_morpheus`*):*

In this framework, as shown in Figure 7, the platform-independent control correctives produced by the control engine are converted to actuator commands of a given vehicle by a platform-dependent MOOS application; for instance, in the *Morpheus* AUV, this is carried out by `pActuatorMap_morpheus`. As outlined in Section II-B, the *Morpheus* class vehicles are equipped with four independently actuated stern control surfaces (i.e. upper rudder, lower rudder, port elevator and starboard elevator), two vertical forward morphing fins and a propeller at the stern end.

The heading and pitch correctives are first mapped out to corresponding stern rudder and elevator angles. If the

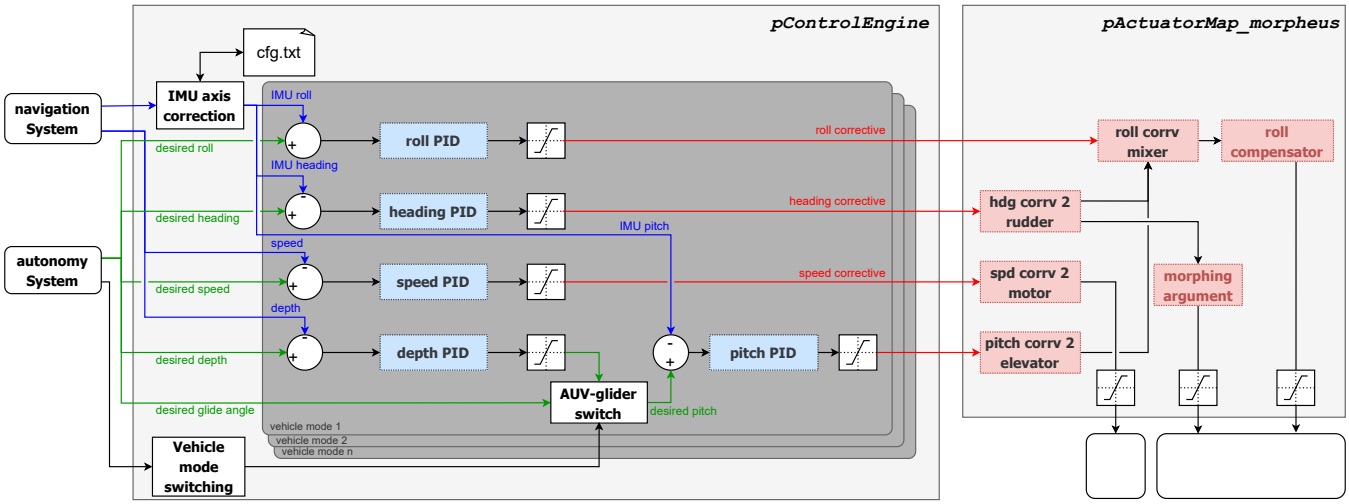


Fig. 7. The low-level control system of *MITFrontseat* has been semi-generalized by sub-dividing it to three components: (1) a platform-independent control engine (`pControlEngine`) that produces control correctives in roll, heading, speed, and pitch; (2) a platform-dependant actuator mapper application (e.g. `pActuatorMap_morpheus`) that maps the control engine outputs to the actuator configuration of a specific vehicle; and (3) actuator drivers that produces low-level signals such as PWM, GPIO and CAN bus messages that drive the actuators.

vehicle is at zero roll angle (or if the roll control subsystem is deactivated), these commands will be the final rudder and elevator commands. However, in situations where the vehicle is rolled, as shown in Figure 8, the zero position of all four stern control surfaces will be offset by a small angle (the maximum deflection limit is typically configured as 5 degrees), attempting to create a righting moment to zero out the roll angle.

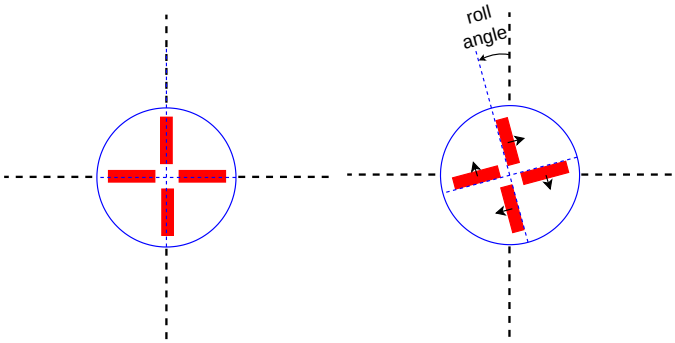


Fig. 8. Roll correction – Left: when the AUV is at zero roll, the control surfaces are at their neutral positions. Right: when the vehicle is rolled, the zero position of all four control surfaces are offset by a small angle, creating a righting moment to correct the vehicle roll angle back to zero.

As shown in Figure 9, when the vehicle is at non-zero roll angles, a rudder deflection will not only create a heading change, but will also create an unintended pitching moment, and vice versa. Thus, the vehicle will have unintended depth fluctuations during turns, and heading fluctuations during depth changes. The roll compensation system within `pActuatorMap_morpheus` attempts to mitigate this by accordingly deflecting the opposing control surfaces to cancel out the unintended moment as given in Equations 6 - 9; for instance, deflecting the elevators to cancel out the unintended pitching moment created by the rudders.

$$uppr\_rudd = \psi^{corr} \cos\phi - \theta^{corr} \sin\phi + \phi^{corr} \quad (6)$$

$$lowr\_rudd = \psi^{corr} \cos\phi - \theta^{corr} \sin\phi - \phi^{corr} \quad (7)$$

$$port\_elev = \psi^{corr} \sin\phi + \theta^{corr} \cos\phi - \phi^{corr} \quad (8)$$

$$stbd\_elev = \psi^{corr} \sin\phi + \theta^{corr} \cos\phi + \phi^{corr} \quad (9)$$

where, $\phi^{corr}$, $\theta^{corr}$ and $\psi^{corr}$ are roll, pitch and heading correctives, and $\phi$, $\theta$ and $\psi$ are roll, pitch and heading angles of the vehicle, respectively. Note that equations 6 - 9 assume that all four stern control surfaces are equal in size and shape.

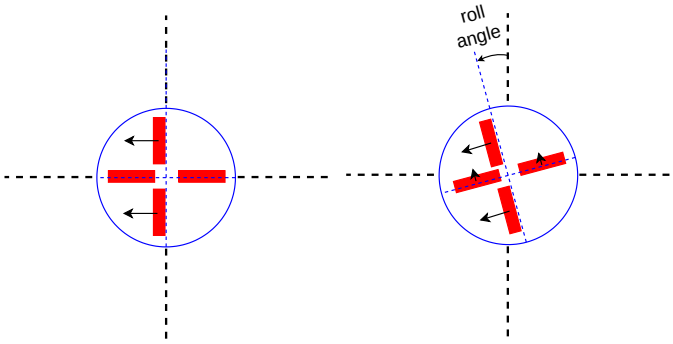


Fig. 9. Roll compensation – Left: when the AUV is at zero roll, heading correction is simply mapped out to a rudder deflection. Right: when the AUV is rolled, however, a simple rudder deflection will not only create a heading change, but also will create an unintended pitch change. Roll compensating system will attempt to mitigate this by accordingly deflecting the elevators to cancel out the pitching moment created by the rudders, and vice versa.

In pActuatorMap_morpheus, the forward located morphing fins are controlled according to the magnitude of the heading error. The fins were deployed if the heading error (i.e. the difference between the desired and current vehicle heading) is larger than 30°. Once deployed, the fins were actively controlled with an equal but opposite angle to the rudder deflection. When the heading error reduced to less than 5°, the morphing fins were retracted.

All final control surface angles are finally published to the MOOSDB as angle as well as normalized commands, which are to be read by the actuator drivers. The speed correctives are also mapped out and published as percentage thrust and normalized thrust commands.

### H. Actuator software drivers

A set of MOOS drivers were developed to communicate with various hardware actuators of the vehicle via hardware interfaces available onboard the BeagleBoard computer. Each driver reads corresponding commands from the MOOSDB, and drives the hardware by providing relevant GPIO, PWM and $I^2C$ commands.

*1) Main motor driver (*iProp*):*

The main motor MOOS driver handles main motor and its related circuitry. Main motor propeller is a hazardous sub-system; hence is protected by an electrical gate that needs to be triggered in order to switch the propeller on. During the mission envelope (which is dictated by pFrontseatManager), The main motor driver triggers the gate by sending a GPIO signal. Subsequently, the relevant PMW signal is sent to the motor, according to the percentage thrust commanded by pActuatorMap_morpheus.

*2) Servo motor driver (*iServo*):*

The servo MOOS driver is responsible for driving the servo motors to the positions commanded by pActuatorMap_morpheus. Servo driver achieves this by providing PWM signals (i.e. via the BeagleBoard's PWM channels) that incrementally changes the servo position until it arrives to the commanded position.

*3) LED strobe driver (*iLED*):*

The LED MOOS driver handles the circuitry related to the vehicle's mast LED strobe. In this framework, the pFrontseatManager posts various different LED pattern commands, each corresponding to the current mode of the vehicle. For instance, four different LED blinking patterns were configured to indicate: (1) a mission has been launched and waiting till the actuator-engage-time, (2) the mission clock is within 10-seconds to the actuator-engage-time, (3) mission is currently being executed and actuators are engaged, and (4) mission has ended and actuators are secured. The LED driver reads these LED commands and sends corresponding GPIO signals to the LED driving circuitry.

### I. Software extension for additional payloads and payload autonomy systems (iMITFrontseat_Gateway)

The base AUVs are typically extended with additional payloads according to its application [74], [75]. To ensure this extendability, the hardware as well as the software of the base vehicle should include boilerplate hooks to interface with additional payload sensors, actuators and processes [74].

The iMITFrontseat_Gateway is a such boilerplate hook that allows payloads (i.e. including payload autonomy systems) to connect to *MITFrontseat* and exchange information. Similar to the iHydroMAN_Gateway discussed in Section III-D.1, this application creates a TCP server, which allows payload systems to connect as a TCP client, and exchange MOOS messages wrapped around a google protocol buffer based standardized message definition. This interface allows payloads to read and publish any MOOS variable to the *MITFrontseat* MOOS community. The standardized message definition and TCP server-client architecture ensures the independence of payload systems; i.e., the payload system does not necessarily need to be a MOOS based system. Multiple payload systems can be connected to *MITFrontseat* at a given instance by spawning multiple instances of iMITFrontseat_Gateway application.

## IV. Morphing fin payload design

The stability and maneuverability indices of a torpedo-shaped vehicle can be dynamically altered using different modes of retractable fin implementations [11]. In this work, we implemented forward located morphing fins, where the stability of an originally stable vehicle can be decreased by deploying the fins; increasing the maneuverability, similar to tuna's dorsal fins. As shown in Figure 10A, the morphing fins were usually retracted during straight runs in order to increase the stability. When the vehicle is required to make a quick heading change, the morphing fins were deployed, as shown in 10B, to destabilize the body, increasing the maneuverability. In addition, the morphing fins were able to be articulated, as shown in Figure 10C, providing a turning moment, to further increase the turning rate. The theoretical derivations of stability-maneuverability criteria, and the mathematical representation of forward-located morphing fins were presented in-detail, in our prior work [11]; therefore, a concise summary is given here in Appendix A.

The morphing payload module was developed as an independent section, that can be outfitted to any place within the mid-body of the base vehicle. Our previous work [11] investigated the variation of the stability index with the location of morphing fins; concluding that a larger stability index variation can be achieved when the fins were located closer to the nose-tip of the vehicle. Thus, in both the *Morpheus* and *Perseus* vehicles, we placed the morphing fin payload module immediately after the nose-cone.

### A. Morphing fin hardware design

The morphing fin hardware design, as shown in Figures 10D - 10F, consists of two morphing fins that were driven in and out of the hull through fin cutouts by push rods. The push rods and their mounting arms were in turn driven by a 32 pitch gearwheel and an oil-filled micro-servo. The fins and rods moved in unison, providing symmetric deployment and retraction. The range of the fin movement was such that when fully retracted, just a few millimeters of fin protrudes from

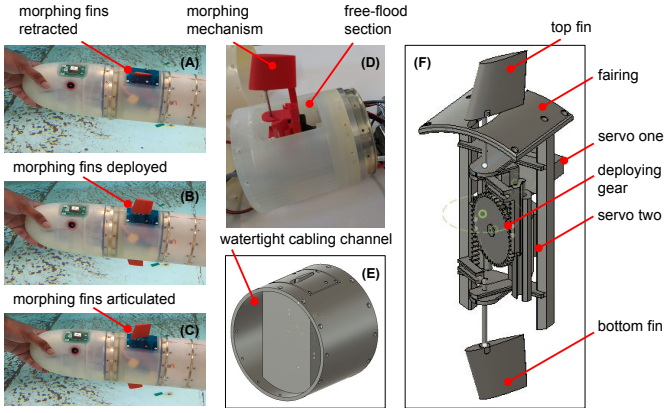


Fig. 10. The morphing fin payload module was placed immediately after the nose-cone of the AUV in order to obtain the maximum stability-index variation. The morphing fins can be (A) retracted, (B) deployed and (C) articulated up to a 20 degree angle of attack. (D) The morphing mechanism was housed inside a free-flood chamber within the module. (E) Two watertight channels were located on either sides of the chamber to run electrical cables across the morphing fin module. (F) The two morphing fins were driven in and out of the hull through fin cutouts by a servo-driven push rods mechanism, which was placed on a carriage that can be rotated using another servo, providing fin articulation.

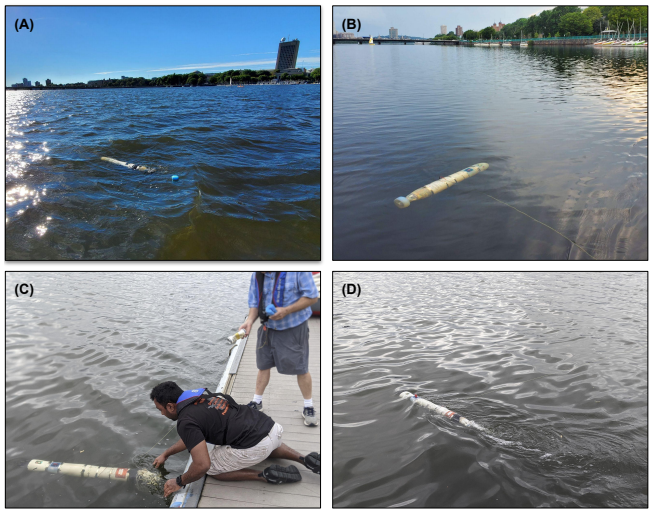


Fig. 11. In-water deployments were conducted in the Charles river, Massachusetts, USA, adjacent to the MIT Sailing Pavilion. Field deployment photos of (A) *MIT-EMATT* AUV, (B) *Morpheus* AUV, and (C-D) *Perseus* AUV.

the hull, while when fully deployed the fin bottom clears the hull, allowing for articulation.

The deploying mechanism was mounted on a carriage with ability to swing approximately 20 degrees to either side, thereby resulting in fin articulation. A 3D-printed rack on the carriage was driven by a 32 pitch gearwheel and an oil-filled articulation micro-servo. Similar to deployment action, the articulation of the fins was also symmetric.

The morphing mechanism was nested into a free-flood hull chamber. Watertight channels were designed on either sides of the chamber, providing watertight wiring channels to run electrical cables across the morphing fin module.

### B. Morphing fin software design

As discussed in Section III-G.2 and illustrated in Figure 7, the platform-independent control correctives produced by the `pControlEngine` were converted to actuator commands of the *Morpheus* vehicle in `pActuatorMap_morpheus`. The adaptive morphing argument was also embedded within this application.

The morphing fins were controlled according to the magnitude of the heading error. The fins were deployed if the heading error (i.e. the difference between the desired and current vehicle heading) is larger than 30°. Once deployed, the fins were actively controlled with an equal but opposite angle to the rudder deflection. When the heading error reduced to less than 5°, the morphing fins were retracted.

## V. Results and Discussion

The original *MIT-EMATT* base vehicle, the optimized base vehicle and *Morpheus* AUV were all extensively field tested in-water in the Charles river, Massachusetts, USA by conducting hundreds of hours of operations over a period of two years (see Figure 11). In this section, we present in-water test results from a set of randomly picked missions.

The vehicle tracks shown in this section were produced using the HydroMAN navigation solution. The base vehicles and *Morpheus* AUV were limited to a depth sensor and an IMU. Therefore, the HydroMAN navigation engine was heavily relying on its embedded vehicle flight dynamic model. The HydroMAN navigation engine requires an initial vehicle motion response dataset to identify the parameters of the vehicle flight dynamic model [51], [56]. In this work, we used the *Perseus* vehicle configuration (shown in Figure 1E), which was outfitted with the piUSBL payload, to obtain the parameter estimation dataset. The piUSBL system was configured as a long baseline (LBL) system, and followed the same methodology as [56] to estimate the vehicle flight dynamic model. Figure 12 compares the HydroMAN navigation solution against the LBL-based navigation solution, for validation and verification purposes.

### A. In-water tests of the MIT-EMATT base vehicle

Figure 13 shows a typical control response plot of the original *MIT-EMATT* base vehicle from an example zig-zag mission. The top subplot shows the desired and actual heading responses of the vehicle together with the rudder commands. As discussed in Section II-B, the original *MIT-EMATT* base vehicle tail-cone had a solenoid-driven rudder and elevator that only allowed bang-bang control. Therefore, as seen from Figure 13 top subplot, the rudder commands had only three positions; i.e. hard-to-port, hard-to-starboard and neutral. This resulted in around 5-10 degree amplitude oscillations in the heading response.

Figure 13 middle subplot shows the desired and actual pitch responses of the vehicle, together with bang-bang elevator commands. As discussed in Section III-G.1, the desired pitch was computed by the depth control loop within `pControlEngine`; attempting to maintain the vehicle depth at the desired depth command. A constant roll angle

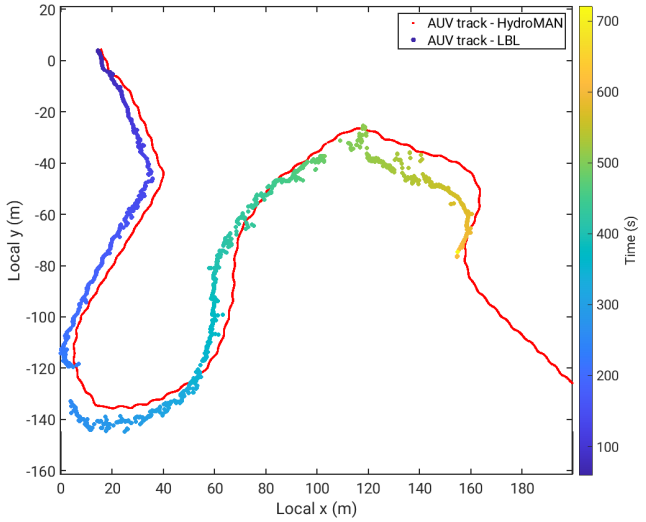


Fig. 12. Comparison of the HydroMAN navigation solution, which was limited to the depth sensor, IMU and the embedded vehicle flight dynamic model, against the navigation solution obtained from an LBL system.

of around 20 degrees was generally observed; primarily as a result of propeller torque. The original *MIT-EMATT* did not have split rudders or split elevators that allowed implementation of active roll control. We addressed this drawback in the optimized vehicle by having individually controlled split rudders and split elevators. In addition, we also included fixed fins with a 3-degree constant angle of attack to counteract the rolling effect due to propeller torque.

Figure 13 bottom subplot illustrates the desired and actual depth responses of the vehicle, which had an oscillation of around 0.2 - 0.4 m amplitude. This is primarily due to the band-bang control strategy and external disturbances.

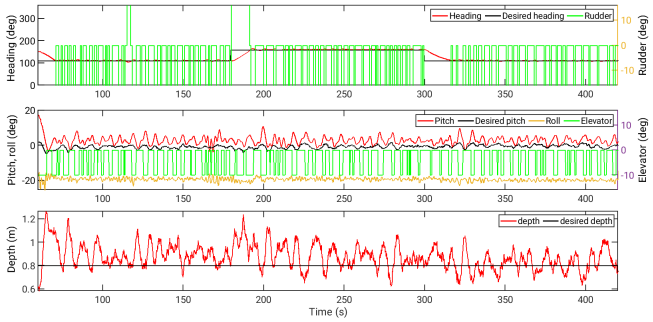


Fig. 13. A control response plot from one of the PID tuning runs conducted with the original *MIT-EMATT* base vehicle, with a solenoid-driven, bang-bang controlled rudder and elevator. The top subplot shows the desired and actual heading responses of the vehicle together with bang-bang rudder commands. Middle plot shows the desired pitch (i.e. the output produced by the depth control loop) and actual pitch responses with bang-bang elevator commands. The roll response is also shown. The bottom plot illustrates the desired and actual depth responses.

Figure 14 shows the vehicle navigation tracks of the original *MIT-EMATT* base vehicle from three identical zig-zag missions, conducted at different thrust percentage values for PID tuning. These missions were conducted using `pHelmPassive`, which publishes pre-scripted, time triggered desired heading and desired depth commands. Such simplified missions were used during PID tuning and heading performance evaluation stages.

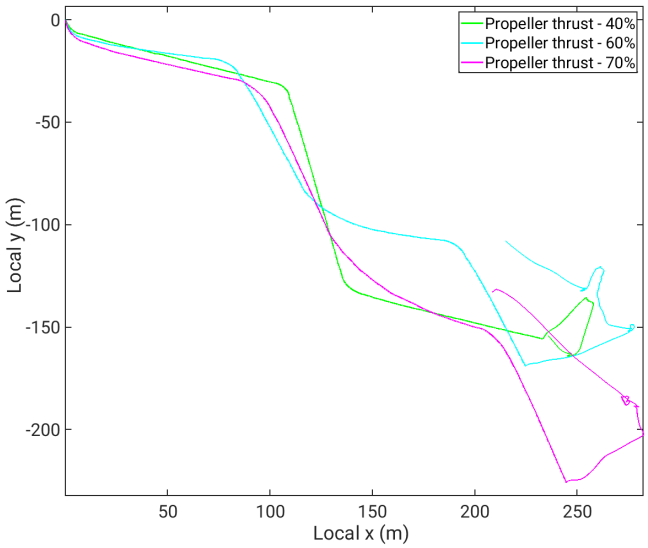


Fig. 14. Tracks of the original *MIT-EMATT* base vehicle conducting three identical zig-zag pattern missions using `pHelmPassive` at various propeller thrust percentages.

### B. In-water tests of the Morpheus AUV

Similar to the *MIT-EMATT* base vehicle, both the optimized base vehicle and *Morpheus* AUV were intensively tested in-water for PID tuning and maneuverability-stability evaluations. Figure 15 shows the vehicle navigation track of the *Morpheus* AUV for two identical zig-zag missions; one with morphing fins engaging according to the argument discussed in Section IV-B, and the second without engaging morphing fins. As seen, both runs provide small turning radii, with the run that engaged morphing fins outperforming the other.

Figure 16 illustrates a comparison of the starboard turns of the same runs. In addition, it also includes a similar turn of the original *MIT-EMATT* base vehicle. The *Morpheus* vehicle provided a turning radius of around 2.5 m when morphing fins were not engaged. The morphing fins were able to further reduce the turning radius down to approximately 1.5 m. In comparison, the turning radius of the original *MIT-EMATT* vehicle was limited to around 10 m. As seen, a significant turning rate improvement was obtained through the use of morphing fins.

Figure 17 shows the turning rate responses of the *Morpheus* AUV for six different example runs, both with and without engaging morphing fins. When comparing top and bottom subplots, the starboard turns always had a significantly higher turning rate as compared to port turn (i.e. approximately 10 deg $s^{-1}$ higher). We believe that this was as a result of the propeller torque favoring the starboard turns.

As seen from Figure 17, the *Morpheus* AUV was able to showcase an exceptional turning rate of around 25-35 deg

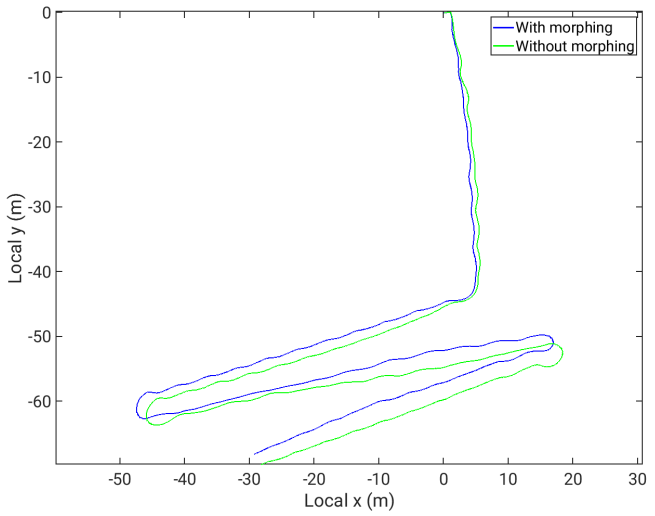


Fig. 15. Tracks of the *Morpheus* AUV conducting two identical zig-zag missions using `pHelmPassive`, with and without employing morphing fins.

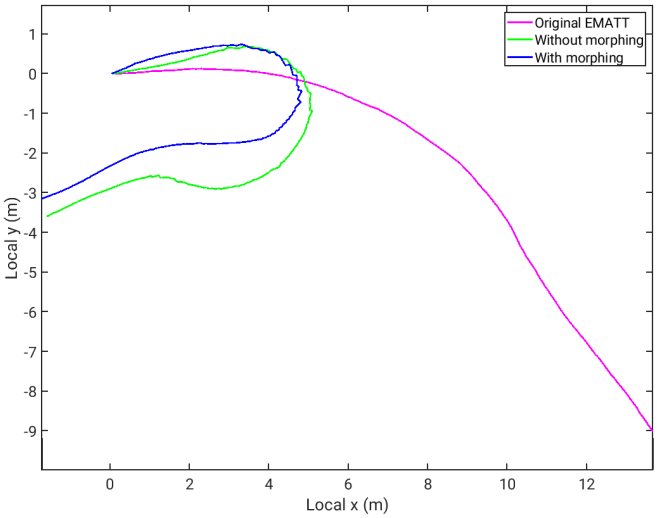


Fig. 16. A visual comparison of the turning radii between the original *MIT-EMATT* vehicle (i.e around 10 m), *Morpheus* without engaging morphing fins (i.e around 2.5 m), and *Morpheus* with morphing fins (i.e around 1.5 m).

$s^{-1}$. A maximum turn rate improvement of around 35% - 50% was gained through the use of morphing fins.

## VI. Conclusions

We designed and constructed an A-sized base AUV, augmented with a stack of modular and extendable hardware and software, including navigation, autonomy, control and high fidelity simulation capabilities. The base vehicle developed in this work was a derivation of the EMATT vehicle hullform, designed and produced by Lockheed Martin Corporation. During the first iteration, we used the original EMATT shell, including the original nose-cone, main-motor bay, and a free-flood tail-cone with solenoid-driven control surfaces; augmented with our own electronics and software stacks. In the second iteration of the base-vehicle, we hydrodynamically

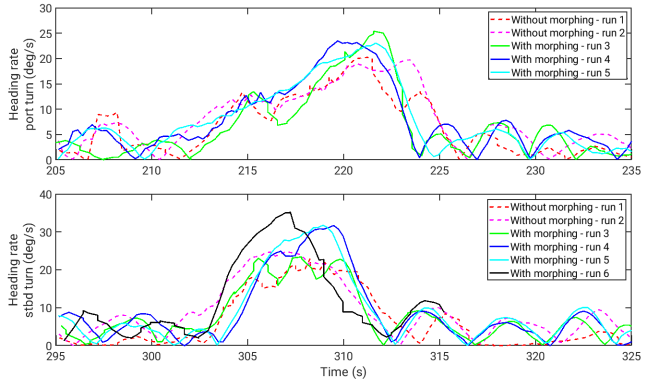


Fig. 17. The heading rate responses of the *Morpheus* AUV observed during a set of randomly picked (upper) starboard and (lower) port turns; with and without engaging morphing fins.

optimized nose and tail cones. The optimized nose-cone included an embedded GPS antenna, LED strobes, external pressure sensor and vacuum port; and the optimized tail-cone included four individually controlled, servo-based control surfaces.

Subsequently, we extended the optimized base vehicle with a novel tuna-inspired morphing fin payload module (referred to as the Morpheus AUV), to achieve good directional stability and exceptional maneuverability; properties that are highly desirable for rigid hull AUVs, but are presently difficult to achieve because they impose contradictory requirements. The morphing fin payload allows the base AUV to dynamically change its stability-maneuverability qualities by using morphing fins, which can be deployed, deflected and retracted, as needed.

The original *MIT-EMATT* base vehicle, the optimized base vehicle and *Morpheus* AUV were all extensively field tested in-water in the Charles river, Massachusetts, USA by conducting hundreds of hours of operations over a period of two years. The *Morpheus* vehicle provided a turning radius of around 2.5 m when morphing fins were not engaged. The morphing fins were able to further reduce the turning radius down to approximately 1.5 m. In comparison, the turning radius of the original *MIT-EMATT* vehicle was limited to around 10 m. The *Morpheus* AUV was able to showcase an exceptional turning rate of around 25-35 deg $s^{-1}$. A maximum turn rate improvement of around 35% - 50% was gained through the use of morphing fins.

## Acknowledgment

This work was funded by a grant from Lockheed Martin Corporation; equipment were provided by the Naval Undersea Warfare Center (NUWC).

We greatly acknowledge Sekhar Tangirala, Stephen Bethel Jr., and Philip Nicolescu at Lockheed Martin Corporation, and Russell Sylvia (now at NUWC) for their technical discussions and continued support for this work. We also acknowledge Prof. Henrik Schmidt at MIT for his autonomy mission manager concept that was adapted to our system; and developers of open source libraries that were used in

this work: Dr. Toby Schneider at gobysoft for NETSIM TCP library; Pierce Nichols for pBBBlue MOOS application; and Strawson Design for the BeagleBone Robotics Cape and robot control library. Many thanks to Emily Mellin, Nikolai Gershfeld, Suparnamaaya Prasad, Tyler Paine and Mike DeFilippo at MIT Sea Grant AUV Laboratory, Dr. Nick Rypkema at WHOI and MIT Sailing Pavilion staff for their support in field trials.

## A. Appendix – Mathematical representation of morphing fins

Utilizing the hydrodynamic coefficient representation of vehicle motion, Triantafyllou et al. [9] provides the theoretical derivation of the stability criterion for underwater vehicles, and how the stability-maneuverability is affected by the stern control surfaces and forward morphing fins. In this section, we summarize the theory given in Triantafyllou et al. [9], and further extend the derivation to show how the stability-maneuverability is affected by the size of stern control surfaces, forward morphing fins, and shroud. The body-fixed axes system and notations utilized in this article is sh

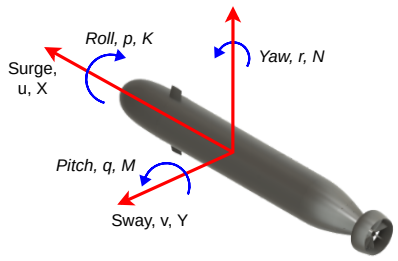


Fig. 18. The body-fixed reference frame used in this article. $X$, $Y$, $Z$ and $K$, $M$, $N$ are the body-fixed forces and moments along/around the surge, sway and heave axes of the AUV, respectively. The linear and angular velocities along/around the surge, sway and heave axes are *u*, *v*, *w* and *p*, *q*, *r*.

### A. Directional stability versus maneuverability

Utilizing the hydrodynamic coefficient representation of vehicle motion [76], the linearized equations of sway and yaw motion of a torpedo-shaped vehicle, decoupled from surge, heave, roll and pitch motion, can be written as given in Equations 10 and 11.

$$Y = (m - Y_{\dot{v}})\dot{v} + (mx_G - Y_{\dot{r}})\dot{r} - Y_v v + (mU - Y_r)r \quad (10)$$

$$N = (I_{zz} - N_{\dot{r}})\dot{r} + (mx_G - N_{\dot{v}})\dot{v} - N_v v + (mx_G U - N_r)r \quad (11)$$

where, $m$ is the mass, $I_{zz}$ is the moment of inertia about the origin, $x_G$ is the longitudinal location of the center of gravity, $U$ is the forward speed of the vehicle, $v$ and $\dot{v}$ are the sway velocity and acceleration, $r$ and $\dot{r}$ are the yaw velocity and acceleration. The hydrodynamic coefficients $-Y_{\dot{v}}$ and $-Y_{\dot{r}}$ denote the added mass in sway due to swaying and yawing acceleration, respectively. $-N_{\dot{v}}$ and $-N_{\dot{r}}$ denote the added moment of inertia due to sway and yaw acceleration. $Y_v$ and $Y_r$ are the linear resistance force in sway due to sway and yaw velocities, and $N_v$ and $N_r$ are the linear resistance moments in yaw due to sway and yaw velocities.

When the rudder is deflected to an angle $\delta$, after the transients die down and a steady turning at forward velocity U, yaw rate r, and side velocity v is achieved, the acceleration terms can be dropped. Then, Equations 10 and 11 become Equations 12 and 13, respectively:

$$-Y_v v + (mU - Y_r)r = Y_\delta \delta \quad (12)$$

$$-N_v v + (mx_G U - N_r)r = N_\delta \delta \quad (13)$$

where, $Y_\delta$ and $N_\delta$ are the linear hydrodynamic coefficients of the rudder. Note that the rudder forces are taken to be a linear function of the rudder angle within this section.

Thus, the yaw rate, $r$ can be written as:

$$r = \frac{Y_\delta \delta}{C}(N_v Y_\delta - Y_v N_\delta) \quad (14)$$

where, the denominator $C$ can be shown to be the *dynamic stability index*, $C$, as given in Equation 15 [76].

$$C = -Y_v(mx_G U - N_r) + N_v(mU - Y_r) \quad (15)$$

If $C > 0$, the body is directionally stable, otherwise, it is linearly unstable. Equation 15 can be recasted as:

$$C = -Y_v(mU - Y_r)(x_r - x_{AC}) \quad (16)$$

where,

$$x_r = \frac{mx_G U - N_r}{mU - Y_r} \quad (17)$$

$$x_{AC} = \frac{N_v}{Y_v} \quad (18)$$

$x_r$ is the distance of the *Center of Rotational motion* (CR) from the origin, i.e. the location where the side force acts when the body performs a pure rotation at constant speed $U$ and (small) angular velocity $r$, and $v = 0$. $x_{AC}$ is the distance from the origin to the *Aerodynamic Center* (AC), i.e. the location where the side force acts when the body performs a steady translation at forward velocity $U$ and side velocity $v$, while $r = 0$ (what is referred to, also, as sideslip velocity).

As noted by [9] [9], the aerodynamic center is a critical quantity in determining the body stability. Since $Y_v$ is always a negative quantity [76], and $(mU - Y_r) > 0$, as $mU$ is a large positive quantity, the stability criterion can be recast as:

$$x_r > x_{AC} \quad (19)$$

As the difference between these two values increases, the linear stability of the vehicle increases while the maneuverability decreases as shown from equation (14).

### B. Presence of stern control surfaces

The effects of the rudder are next added to the equations of motion, with the subscript *b* corresponding to the bare body coefficients. Following [9] [9], the updated hydrodynamic coefficients are:

$$Y_v = Y_{v,b} + \frac{Y_\delta}{U} \tag{20}$$

$$Y_r = Y_{r,b} - x_R \frac{Y_\delta}{U} \tag{21}$$

$$N_v = N_{v,b} - x_R \frac{Y_\delta}{U} \tag{22}$$

$$N_r = N_{r,b} - x_R^2 \frac{Y_\delta}{U} \tag{23}$$

where $x_R = N_\delta / Y_\delta$ is the location where the force acts on the rudder and $Y_\delta$ and $N_\delta$ are defined earlier.

The stability index is now updated to take into account the effect of the rudder. Plugging the updated hydrodynamic coefficients into Equation (15) and denoting the stability index of just the bare body by $C_b$ leads to the following updated stability index:

$$C = C_b - A(mx_G U - N_{r,b} - Y_{v,b}\xi^2 - N_{v,b}\xi + (mU - Y_{r,b})\xi) \tag{24}$$

which reduces to:

$$C = C_b - A[-Y_{v,b}\xi(x_{AC,b}+\xi) + (mU - Y_{r,b})(x_{r,b}+\xi)] \tag{25}$$

where

$$A = \frac{Y_\delta}{U} = \frac{L_{rudder}}{U} < 0 \tag{26}$$

$$\xi = -x_R > 0 \tag{27}$$

The yaw rate from Equation (14) is calculated within linear theory as:

$$\frac{r}{\delta} = \frac{A}{CU} Y_{v,b}(x_{AC,b} + \xi) \tag{28}$$

### C. The size of stern control surfaces

The lift generated by the stern control surfaces contributes to the hydrodynamic coefficients, as shown in Equations 20-23. To estimate how much lift is generated by the rudder, the general form for lift is used:

$$L_{rudder} = \frac{1}{2}\rho C_L (2S_{rudder}) U^2 \tag{29}$$

where $C_L$ is the coefficient of lift and $S_{rudder}$ is the area of one of the two rudders.

As [9] [9] concluded, the addition of these stern control surfaces can stabilize an initially unstable vehicle, as long as it is above a threshold value that provides stability. Rearranging Equation 25 determines what this threshold value for *A* should be in order to bring the stability index $C$ from a negative to a positive value. Since *A* is determined by the amount of lift generated for a certain speed, the size of the rudder is what keeps this value close to the threshold value. If the size of the rudder increases, resulting in an increase in *A*, the vehicle surpasses the stability threshold, becoming more stable and reducing the turning rate of the vehicle. The value of *C* should remain close to this stability transition in order for the rudder to have a significant effect on the turning rate.

### D. The presence of forward morphing fins

The addition of forward morphing fins to an underwater vehicle has also an effect on the stability and maneuverability of the vehicle. The hydrodynamic coefficients from Equations 20-23 are updated to take into account the forward morphing fin following a similar process used for the rudder [9]:

$$Y_v = Y_{v,b} + \frac{Y_\delta}{U} + \frac{Y_{f\delta}}{U} \tag{30}$$

$$Y_r = Y_{r,b} - x_R \frac{Y_\delta}{U} - x_f \frac{Y_{f\delta}}{U} \tag{31}$$

$$N_v = N_{v,b} - x_R \frac{Y_\delta}{U} - x_f \frac{Y_{f\delta}}{U} \tag{32}$$

$$N_r = N_{r,b} - x_R^2 \frac{Y_\delta}{U} - x_f^2 \frac{Y_{f\delta}}{U} \tag{33}$$

where $x_f = N_{f\delta}/Y_{f\delta}$ is the location where the force acts on the vehicle, $Y_{f\delta}$ is the fin force per unit rudder angle and $N_{f\delta}$ is the moment per unit angle. Once again, the stability index in Equation 25 is updated to reflect the effects of the forward morphing fin:

$$\begin{aligned} C = C_b - \quad & A[(x_r+\xi)(mU - Y_{r,b}) - Y_{v,b}\xi(x_{AC}+\xi)] \\ & -B[(x_r-\eta)(mU - Y_{r,b}) - Y_{v,b}\eta(\eta - x_{AC})] \\ & +2AB(\eta^2 + \xi^2) \end{aligned} \tag{34}$$

with $B = \frac{Y_{f\delta}}{U} < 0$ and $\eta = x_f$.

Hence, the yaw rate in Equation 28 can be calculated with the updated stability index, *C*, that takes into account the effects of both the rudder and forward morphing fin. The values of $mU - Y_{r,b} > 0$ and $-Y_{v,b} > 0$ require that $\eta > x_r$ and $\eta < x_{AC}$. In other words, the forward morphing fin needs to be positioned ahead of the bare body center of rotational motion and behind the bare body aerodynamic center of the vehicle, requiring that the bare body vehicle be initially directionally unstable. For a given stable vehicle and rudder configuration, the main way to increase the turning rate is to decrease the stability parameter.

### E. The size and angle-of-attack of forward morphing fins

A fish bends its body when it initiates a turn [9]. Since the forward morphing fins are located ahead of the center of gravity, the fins deflect in the opposite direction of the rudder. This can be adapted to the coefficients derived so far for the case when $\delta_f = -\delta$, the magnitude of the forward morphing fin and rudder angles are equal but in opposite directions. The deflection does not affect the stability criterion. The turning rate for a vehicle with both rudder and forward morphing fins is:

$$\frac{r}{\delta} = \frac{1}{CU}[(-A)(-Y_{v,b})(x_{AC}+\xi) + (-B)(-Y_{v,b})(\eta - x_{AC}) + 2(-A)(-B)(\eta+\xi)] \quad (35)$$

where $\eta = x_f$ is the fin position, $\xi = -x_R$ is the rudder position, $A = \frac{Y_\delta}{U}$ and $B = \frac{Y_{f\delta}}{U}$. The last term in the equation contributes the strongest increase in the turning rate since $-A > 0$, $-B > 0$, and $-Y_{v.b} > 0$ [9]. In order to increase the rate of turning of the vehicle, the forward morphing fins should have a comparable size and lift generation as the rudder.

## REFERENCES


[1] H. Eda, "Directional stability and control of ships in waves," *Journal of Ship Research*, vol. 16, no. 03, pp. 205–218, 1972.

[2] N. Minorsky, "Directional stability of automatically steered bodies," *Journal of the American Society for Naval Engineers*, vol. 34, no. 2, pp. 280–309, 1922.

[3] A. Maki, T. Tsutsumoto, and Y. Miyauchi, "Fundamental research on the maneuverability of the underwater vehicle having thrust vectoring system," *Journal of Marine Science and Technology*, vol. 23, no. 3, pp. 495–506, 2018.

[4] K. R. Armo, "The relationship between a submarine's maximum speed and its evasive capability," The Naval Postgraduate School, Monterey, California, Tech. Rep., 2000.

[5] A. Jones, "The effect of submarine maximum speed on the hit probability of an air-launched torpedo," The Admiralty Underwater Weapons Establishment, Portland, Tech. Rep., 1973.

[6] P. R. Bandyopadhyay, "Maneuvering hydrodynamics of fish and small underwater vehicles," *Integrative and comparative biology*, vol. 42, no. 1, pp. 102–117, 2002.

[7] S. Ziaeefard, B. R. Page, A. J. Pinar, and N. Mahmoudian, "A novel roll mechanism to increase maneuverability of autonomous underwater vehicles in shallow water," in *OCEANS 2016 MTS/IEEE Monterey*. IEEE, 2016, pp. 1–5.

[8] K. S. Varyani, P. Krishnankutty, and R. McGregor, "Effect of rudder size and location on the turning performance of a high speed swath ferry," *IFAC Proceedings Volumes*, vol. 36, no. 21, pp. 61–66, 2003.

[9] M. S. Triantafyllou, N. Winey, Y. Trakht, R. Elhassid, and D. Yoerger, "Biomimetic design of dorsal fins for AUVs to enhance maneuverability," *Bioinspiration & biomimetics*, vol. 15, no. 3, p. 035003, 2020.

[10] M. Bettle, "Validating design methods for sizing submarine tailfins," *Proceedings of warship*, 2014.

[11] S. Randeni, E. M. Mellin, M. Sacarny, S. Cheung, M. Benjamin, and M. Triantafyllou, "Bioinspired morphing fins to provide optimal maneuverability, stability, and response to turbulence in rigid hull AUVs," *Bioinspiration & Biomimetics*, vol. 17, no. 3, p. 036012, 2022.

[12] A. Phillips, S. Turnock, and M. Furlong, "The use of computational fluid dynamics to aid cost-effective hydrodynamic design of autonomous underwater vehicles," *Proceedings of the Institution of Mechanical Engineers, Part M: Journal of Engineering for the Maritime Environment*, vol. 224, no. 4, pp. 239–254, 2010.

[13] R. A. Holler, "The evolution of the sonobuoy from world war II to the cold war," Navmar Applied Sciences Corporation, Tech. Rep., 2014.

[14] Lockheed-Martin, "MK39 expendable mobile ASW training target and field programmability system (EMATT)." [Online]. Available: https://www.lockheedmartin.com/en-us/products/a-size-autonomous-underwater-vehicles.html

[15] P. W. Webb, "Form and function in fish swimming," *Scientific American*, vol. 251, no. 1, pp. 72–83, 1984.

[16] F. E. Fish and A. J. Nicastro, "Aquatic turning performance by the whirligig beetle: constraints on maneuverability by a rigid biological system," *Journal of Experimental Biology*, vol. 206, no. 10, pp. 1649–1656, 2003.

[17] X. Li, "Hydrodynamic analysis for the morphing median fins of tuna during yaw motions," *Applied Bionics and Biomechanics*, 2021.

[18] M. S. Triantafyllou, "Tuna fin hydraulics inspire aquatic robotics," *Science*, vol. 357, no. 6348, pp. 251–252, 2017.

[19] F. E. Fish and G. V. Lauder, "Control surfaces of aquatic vertebrates: active and passive design and function," *Journal of Experimental Biology*, vol. 220, no. 23, pp. 4351–4363, 2017.

[20] S. Du, Z. Wu, J. Wang, S. Qi, and J. Yu, "Design and control of a two-motor-actuated tuna-inspired robot system," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 2019.

[21] D. Matthews and G. V. Lauder, "Fin-fin interactions during locomotion in a simplified biomimetic fish model," *Bioinspiration & Biomimetics*, 2021.

[22] P. Han, G. V. Lauder, and H. Dong, "Hydrodynamics of median-fin interactions in fish-like locomotion: Effects of fin shape and movement," *Physics of Fluids*, vol. 32, no. 1, p. 011902, 2020.

[23] C. H. White, G. V. Lauder, and H. Bart-Smith, "Tunabot flex: a tuna-inspired robot with body flexibility improves high-performance swimming," *Bioinspiration & Biomimetics*, vol. 16, no. 2, p. 026019, 2021.

[24] F. E. Fish, "Advantages of aquatic animals as models for bio-inspired drones over present AUV technology," *Bioinspiration & biomimetics*, vol. 15, no. 2, p. 025001, 2020.

[25] A. Phillips, M. Haroutunian, S. Man, A. Murphy, S. Boyd, J. Blake, and G. Griffiths, "Nature in engineering for monitoring the oceans: Comparison of the energetic costs of marine animals and AUVs," 2012.

[26] N. R. Rypkema, E. M. Fischel, and H. Schmidt, "Closed-loop single-beacon passive acoustic navigation for low-cost autonomous underwater vehicles," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 641–648.

[27] N. R. Rypkema, H. Schmidt, and E. M. Fischell, "Synchronous-clock range-angle relative acoustic navigation: A unified approach to multi-auv localization, command, control and coordination," *arXiv preprint arXiv:2110.13825*, 2021.

[28] E. M. Mellin, "Using biomimetics to improve the maneuvering performance of the expendable mobile antisubmarine warfare training target (EMATT)," Ph.D. dissertation, Massachusetts Institute of Technology, 2021.

[29] Blue-Robotics. (2021) Bar30 high-resolution 300m depth/pressure sensor. [Online]. Available: https://bluerobotics.com/store/sensors-sonars-cameras/sensors/bar30-sensor-r1/

[30] TE-Connectivity-Sensors, *MS5837-30BA Ultra-small, gel-filled, pressure sensor with stainless steel cap*, TE-Connectivity-Sensors, 2019.

[31] R. Grimmett, *BeagleBone Robotic Projects*. Packt Publishing Ltd, 2013.

[32] S. Furber and A. Wilson, "The acorn risc machine - an architectural view," *Electronics and Power*, vol. 33, no. 6, pp. 402–405, 1987.

[33] M. Astley, D. C. Sturman, and G. A. Agha, "Middleware," *Communications of the ACM*, vol. 44, no. 5, p. 99, 2001.

[34] N. Mohamed, J. Al-Jaroodi, and I. Jawhar, "Middleware for robotics: A survey," in *2008 IEEE Conference on Robotics, Automation and Mechatronics*. Ieee, 2008, pp. 736–742.

[35] A. Watson, "Omg (object management group) architecture and corba (common object request broker architecture) specification," in *IEE Colloquium on Distributed Object Management*. IET, 1994, pp. 4–1.

[36] P. M. Newman, "MOOS - mission orientated operating suite," 2008.

[37] G. Pardo-Castellote, "Omg data-distribution service: Architectural overview," in *23rd International Conference on Distributed Computing Systems Workshops, 2003. Proceedings.* IEEE, 2003, pp. 200–206.

[38] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Y. Ng, *et al.*, "ROS: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2. Kobe, Japan, 2009, p. 5.

[39] T. Schneider, "Goby3: A new open-source middleware for nested communication on autonomous marine vehicles," in *2016 IEEE/OES Autonomous Underwater Vehicles (AUV)*. IEEE, 2016, pp. 236–240.

[40] A. S. Huang, E. Olson, and D. C. Moore, "LCM: Lightweight communications and marshalling," in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2010, pp. 4057–4062.

[41] Google-Developers. (2022) Protocol buffers. [Online]. Available: https://developers.google.com/protocol-buffers

[42] InvenSense, *MPU-9250 Product Specification*, InvenSense Inc., 2016.

[43] P. Nichols. (2020) Third party software MOOS-IvP extensions - Project Ladon: pBB-Blue. [Online]. Available: https://oceanai.mit.edu/moos-ivp/pmwiki/pmwiki.php?n=Manifest.PBBBlue

[44] Strawson-Design. (2020) Robot control library. [Online]. Available: http://www.strawsondesign.com/docs/librobotcontrol/index.html

[45] Xsens, *MTi 1-series User Manual*, Xsens, 2021.

[46] Adafruit-Industries, *Adafruit FONA 3G Cellular + GPS Breakout*, Adafruit Industries, 2022.

[47] SimCom, *SIM5320 Hardware Design V1.07*, Sim Com, 2012.

[48] GD-MS. (2016) Bluefin uuv dives deep into the arctic at u.s. navy exercise. [Online]. Available: https://gdmissionsystems.com/articles/2016/09/16/news-2016-bluefin-uuv-goes-deep-/into-the-arctic-at-icex-2016

[49] ISE. (2017) International submarine engineering explorer autonomous underwater vehicle. [Online]. Available: https://ise.bc.ca/wp-content/uploads/2017/12/Explorer_SpecSheet.pdf

[50] Teledyne-Gavia. (2018) Teledyne gavia releases ixblue phins compact c3 module. [Online]. Available: http://www.teledynemarine.com/press-releases/teledyne-gavia-releases-ixblue-/phins-compact-c3-module

[51] S. Randeni, T. Schneider, E. Bhatt, O. A. Víquez, and H. Schmidt, "A high-resolution AUV navigation framework with integrated communication and tracking for under-ice deployments," *Journal of Field Robotics*, 2022.

[52] S. Randeni, T. Schneider, and H. Schmidt, "HydroMAN 2.0: An environmentally adaptive, self-learning UUV navigation engine," in *PMS 406 Workshop on Improving Autonomous Systems for Next-Generation UUVs*. United States Navy, March 2022.

[53] S. Randeni, M. Menjamin, M. Triantafyllou, and H. Schmidt, "A software toolkit for rapid development of AUVs – using MOOS-IvP with MITFrontseat, HydroMAN and VECTORS," in *MOOS Development and Applications Working Group*. Massachusetts Institute of Technology, August 2022.

[54] S. Randeni, T. Schneider, and H. Schmidt, "Construction of a high-resolution under-ice AUV navigation framework using a multidisciplinary virtual environment," in *2020 IEEE/OES Autonomous Underwater Vehicles Symposium (AUV)(50043)*. IEEE, 2020, pp. 1–7.

[55] S. Randeni, E. M. Fischell, and H. Schmidt, "An AUV dynamic model, based on the conservation of energy, for underwater navigation aiding," *IEEE Journal of Oceanic Engineering (Under Review)*, 2020.

[56] S. A. T. Randeni, N. R. Rypkema, E. M. Fischell, A. L. Forrest, M. R. Benjamin, and H. Schmidt, "Implementation of a hydrodynamic model-based navigation system for a low-cost AUV fleet," in *IEEE OES Autonomous Underwater Vehicle Symposium*, 2018.

[57] M. G. McPhee, "An analysis of pack ice drift in summer," *Sea ice processes and models*, pp. 62–75, 1980.

[58] C. Kaminski, T. Crees, J. Ferguson, A. Forrest, J. Williams, D. Hopkin, and G. Heard, "12 days under ice–an historic AUV deployment in the canadian high arctic," in *2010 IEEE/OES Autonomous Underwater Vehicles*. IEEE, 2010, pp. 1–11.

[59] T. Mo-Bjørkelund, P. Norgren, and M. Ludvigsen, "Simulation and forecasting of ice drift as a tool for autonomous under ice operations," in *2020 IEEE/OES Autonomous Underwater Vehicles Symposium (AUV)(50043)*. IEEE, 2020, pp. 1–6.

[60] M. R. Benjamin, H. Schmidt, P. M. Newman, and J. J. Leonard, "Nested autonomy for unmanned marine vehicles with moos-ivp," *Journal of Field Robotics*, vol. 27, no. 6, pp. 834–875, 2010.

[61] M. Benjamin. (2022) Moos-ivp home page. [Online]. Available: http://www.moos-ivp.org/

[62] A. Balasuriya, S. Petillo, H. Schmidt, and M. Benjamin, "Behavior-based planning and prosecution architecture for autonomous underwater vehicles in ocean observatories," in *OCEANS'10 IEEE SYDNEY*. IEEE, 2010, pp. 1–5.

[63] D. P. Eickstedt and S. R. Sideleau, "The backseat control architecture for autonomous robotic vehicles: A case study with the iver2 AUV," in *OCEANS 2009*. IEEE, 2009, pp. 1–8.

[64] O. A. Viquez, E. M. Fischell, N. R. Rypkema, and H. Schmidt, "Design of a general autonomy payload for low-cost AUV r&d," in *2016 IEEE/OES Autonomous Underwater Vehicles (AUV)*. IEEE, 2016, pp. 151–155.

[65] J. E. Naglak, B. R. Page, and N. Mahmoudian, "Backseat control of sandshark AUV using ROS on Raspberry-pi," in *OCEANS 2018 MTS/IEEE Charleston*. IEEE, 2018, pp. 1–5.

[66] J. Hwang, N. Bose, G. Millar, A. B. Gillard, H. D. Nguyen, and G. Williams, "Enhancement of AUV autonomy using backseat driver control architecture," *International Journal of Mechanical Engineering and Robotics Research*, vol. 10, no. 6, pp. 292–300, 2020.

[67] A. Underwood and C. Murphy, "Design of a micro-auv for autonomy development and multi-vehicle systems," in *OCEANS 2017-Aberdeen*. IEEE, 2017, pp. 1–6.

[68] T. Schneider and H. Schmidt, "Goby-acomms: A modular acoustic networking framework for shortrange marine vehicle communications," *URL http://gobysoft.com/dl/goby-acomms1.pdf*, 2012.

[69] T. E. Schneider and H. Schmidt, "Goby-acomms version 2: extensible marshalling, queuing, and link layer interfacing for acoustic telemetry," *IFAC Proceedings Volumes*, vol. 45, no. 27, pp. 331–335, 2012.

[70] T. Schneider, *Goby Underwater Autonomy Project: User Manual for Version 3.0.12.*, GobySoft, 2022. [Online]. Available: https://gobysoft.org/dl/goby3-user-manual.pdf

[71] L. Freitag, M. Grund, S. Singh, J. Partan, P. Koski, and K. Ball, "The whoi micro-modem: An acoustic communications and navigation system for multiple platforms," in *Proceedings of OCEANS 2005 MTS/IEEE*. IEEE, 2005, pp. 1086–1092.

[72] T.-H. Won and S.-J. Park, "Design and implementation of an omni-directional underwater acoustic micro-modem based on a low-power micro-controller unit," *Sensors*, vol. 12, no. 2, pp. 2309–2323, 2012.

[73] B.-C. Renner, J. Heitmann, and F. Steinmetz, "ahoi: Inexpensive, low-power communication and localization for underwater sensor networks and $\mu$auvs," *ACM Transactions on Sensor Networks (TOSN)*, vol. 16, no. 2, pp. 1–46, 2020.

[74] P. E. Hagen and J. Kristensen, "The HUGIN AUV "plug and play" payload system," in *OCEANS'02 MTS/IEEE*, vol. 1. IEEE, 2002, pp. 156–161.

[75] R. Camilli, B. Bingham, M. Jakuba, H. Singh, and J. Whelan, "Integrating in-situ chemical sampling with AUV control systems," in *Oceans' 04 MTS/IEEE Techno-Ocean'04 (IEEE Cat. No. 04CH37600)*, vol. 1. IEEE, 2004, pp. 101–109.

[76] M. S. Triantafyllou and F. S. Hover, *Maneuvering and Control of Marine Vehicles*. MIT Press, Cambridge, MA, 2002.