Project Report

ONR BAA Announcement #N00014-19-B001

HURACAN: 21” Long Range Autonomous Underwater Vehicle

Focus on the Distributed Computing and Power Control System

Reviewed Period: April 1, 2020 to April 19, 2023

Final Performance Report

Principal Investigator: Christopher Kitts

Professor of Mechanical Engineering and Associate Dean
Santa Clara University
500 El Camino Real, Santa Clara, CA. 95053
ckitts@scu.edu
408-554-4382

Technical Point of Contact: William Kirkwood
Monterey Bay Aquarium Research Institute
7700 Sandholdt Rd, Moss Landing, CA 95039
kiwi@mbari.org
831-775-1707

Financial Point of Contact: Mary-Ellen Fortini
Santa Clara University
500 El Camino Real
St. Joseph’s Hall, Rm 116, Santa Clara, CA. 95053
mfortini@scu.edu
408-554-4806

Table of Contents

Abstract

The vision of the Huracan program is to explore the development of a new Autonomous Underwater Vehicle (AUV) reference design capable of very long duration autonomous operations, for periods extending for weeks or possibly even more than a month. The development approach was to begin by funding a Phase 1 effort in which a distributed computing and power control system would be designed and prototyped, given the critical role that such a system would play in any feasible design. Possible additional Phases were conceived to address implementation of the high energy capacity power storage system, design of a vehicle control and navigation system, development and fabrication of a hull, field testing and design iteration, and ultimately, vehicle fabrication and integration with a variety of mission-specific instruments and sensors. This document describes work performed for Phase 1 of this project, which was conducted over a three year period from April 2020 - April 2023. In doing so, the document details the design of a highly capable, flexible, and low power computing system.

1.0 Summary of Project Goals and Achievements

The goal of this project was to develop a low-power, flexible, distributed computing and power control backbone for a novel class of long-duration, dwell-capable, 21" diameter Autonomous Underwater Vehicles (AUVs). A specific objective was to develop a system capable of supporting significantly longer missions than existing AUVs owned and operated by the Monterey Bay Aquarium Research Institute (MBARI). This program was envisioned as the first phase of a possibly multi-phase program in developing a full prototype of the envisioned AUV class, currently nicknamed HURACAN.

Rather than adapt an existing MBARI computing and power control system (which was more than a decade old in terms of design heritage), which was one of the stated design options to be considered, the team developed a completely new, distributed, modernized, embedded control system that uses very low-power electronics, provides high-resolution power configuration control, provides local functional control (for functions such as payload control, navigation tasks, etc.), and which can be used for computing nodes that are on the order of dozens of feet apart. The system also includes low-level control (beyond that provided directly by the manufacturer) of a new breed of high-capacity batteries that enables significantly higher energy density in order to support long range missions. The system also includes a vast array of software for basic operation of computational nodes, for specific functional capabilities within particular nodes (e.g., functions such as heading control, communications, instrument control, and so on), mission sequencing, diagnostic support, etc.

While the distributed system can accommodate any arbitrary computer or device that conforms to the interface specification (CAN communications, power bus, etc.), a custom low-power computing node reference design was developed. This low-power design allows local power and functional control, and it supports connectivity to peripheral devices through a wide range of interface standards/approaches. Furthermore, it is easily adapted to allow portions of the board to be populated or not (to save power/cost by not populating unused peripheral channels), to allow the number of existing channels of any particular type to be increased/reduced, and to even integrate new channels.

In terms of design verification, we have successfully completed several design iterations with functional hardware/software, to include not only protoboarded testbeds but two complete revisions of fabricated printed circuit board (PCB) computing boards. As part of this process, we have demonstrated all desired system functionalities in terms of local power control, interfacing with a variety of peripheral components, local functional control to include real time PID and state machine control functions, communications among nodes, centralized task sequencing, off-board communications, equipment status monitoring, and so on.

In addition, we are installing a simple build of the system into a small test AUV prototype in order to identify and iteratively address issues that aren't apparent until a real field deployment is conducted. With this system we have demonstrated out of water mock deployments that incorporated remote manual joystick control, wireless diagnostic and mission configuration management, surface GPS waypoint navigation, and heading-controlled mow-the-lawn capability. We have also conducted several test tank in-water tests and will soon be performing an open-water surface test of the above-mentioned capabilities. We note that this level of open water testing was not a requirement of the initial phase but was deemed important to help advance refinement of our design.

Figure 1-1: *The MOANA test AUV which has been configured to test the designed distributed computing and power control system.*

1.1 Specific Achievements: Highlights of achievements for this Phase 1 effort include the following:

- Definitions of Huracan-class AUV vehicle and mission concept;
- Definition of a distributed computing architecture capable of high resolution power configuration, local functional control, low-power implementation, and implementation in a physical implementation supporting component distribution over tens of feet;
- Design of a distributed computing interface specification and software libraries to support data communications between computing nodes for command, telemetry, and power control services;

- Design of a new computing node motherboard with local high resolution power configuration, the ability to run local real time functional controllers (for tasks such as heading control, instrument control, etc.), a variety of interface support for external components/peripherals, and the ability to be easily adapted to tailor the number of interface channels and to incorporate new interface specifications;
- Implementation of two revisions of a specific motherboard build for a set of candidate peripheral components and functions;
- Design of power system using a state-of-the-art high energy density battery cell and a custom control interface for power control and integration;
- Development of a bench model implementation of an entire integrated AUV electronics system using the developed distributed computing and power subsystems, simple candidate components for navigation (thruster, rudder servo, mass slide system for pitch control, heading and pitch sensor, GPS, etc.), communications (wireless surface comms), candidate instruments (streaming camera) with a high data rate storage bus/drive, remote surface driving and diagnostic capability, a safety mass drop device, a dead-man switch for testing, and a simple on-board mission scripting capability;
- Demonstration of the system in the context of a simple mission, packaged within a small demonstrator-model AUV shell, shown in Figure 1-1, and commanded to conduct several mission-like functions such as heading control, navigating a planar mow-the-lawn pattern, conducting a waypoint-based go-to function appropriate for surface navigation, etc.; these functions have been successfully demonstrated in the AUV as through simulated propulsion in an outdoor environment and are being prepared for an open water test in early June 2023.

1.2 Challenges Due To Covid: Significant challenges were faced in the execution of this program, requiring a one year no cost extension. These challenges included the covid pandemic and its resulting paralysis of the nation's electronics supply chain. For the first year of the program, Santa Clara University (SCU) and Monterey Bay Aquarium Research Institute (MBARI) labs were effectively closed, such that the team had to work from their various homes with no ability to gather together, limited access to advanced software design tools, and no access to prototyping facilities. SCU's labs began reopening in mid-2022, but MBARI's facility remained largely shut down until early 2023. These closures and their related effects significantly delayed progress in the design and early prototyping phases. SCU also faced temporary hiring freezes/delays, longer times in processing purchase orders / receipts / delivery requests, and so on.

Furthermore, the pandemic crippled the nation's supply chain with a significant impact being the delay of electronic components. This impact was so great that, as an example, entire automobile manufacturing plants were shut down due to the unavailability of parts. For the

components used in this development effort, many availability delays were on the order of months and a few were projected at more than a year; in some cases, no delivery times were even provided, and in other cases once a waiting period neared its end the vendor would inform customers that the part was simply discontinued (after waiting months for the part). While the team created a design somewhat independent of these effects, we were forced to make some modifications by using non-ideal but available parts in order to make progress when prototyping. The ultimate result was that while we have a design that meets the original objectives of the program, implemented prototypes are in some cases “Frankensteins” in the sense that they are functionally equivalent but a) may not be the ideal part, or b) may be a suitable part encased within a development board (which may include components and functions not required). Because of this, the emphasis has been on establishing the functionality required for the HURACAN design while sometimes using less power-efficient parts/assemblies, etc.

2.0 The Huracan System Concept

MBARI's Long Range AUV (LRAUV) program is a development effort that integrates the latest battery technologies with advanced power control and low drag elements in order to provide a vehicle with outstanding mission duration and range. There are a number of reasons to pursue platforms with these characteristics for science, industry and the military. The Huracan concept is based on these same aims, but primarily focused on potential users with applications where duration is a dominant requirement. In particular, Huracan is being developed to be ideal for missions in which extended dwell time in a region of interest may be required in order to detect and react to unpredictable conditions, such as scientific phenomena.

Figure 2-1 shows the concept of operations for the Huracan system. While capable of conventional ship-based deployment, the design supports crane-based deployment from a pier with power-efficient low-speed travel to its region of operation. The AUV will be capable of dwelling on-site for weeks at a time, autonomously monitoring conditions and/or periodically checking for remote commands in order to determine when payload operations are appropriate. The vehicle will accommodate a rich set of payload capabilities, to include mission profiles that include environmental sensing/sampling, physical intervention, and ejection of deployable payload modules. Upon mission completion or when low on power, the system will navigate back to shore for crane-based retrieval.

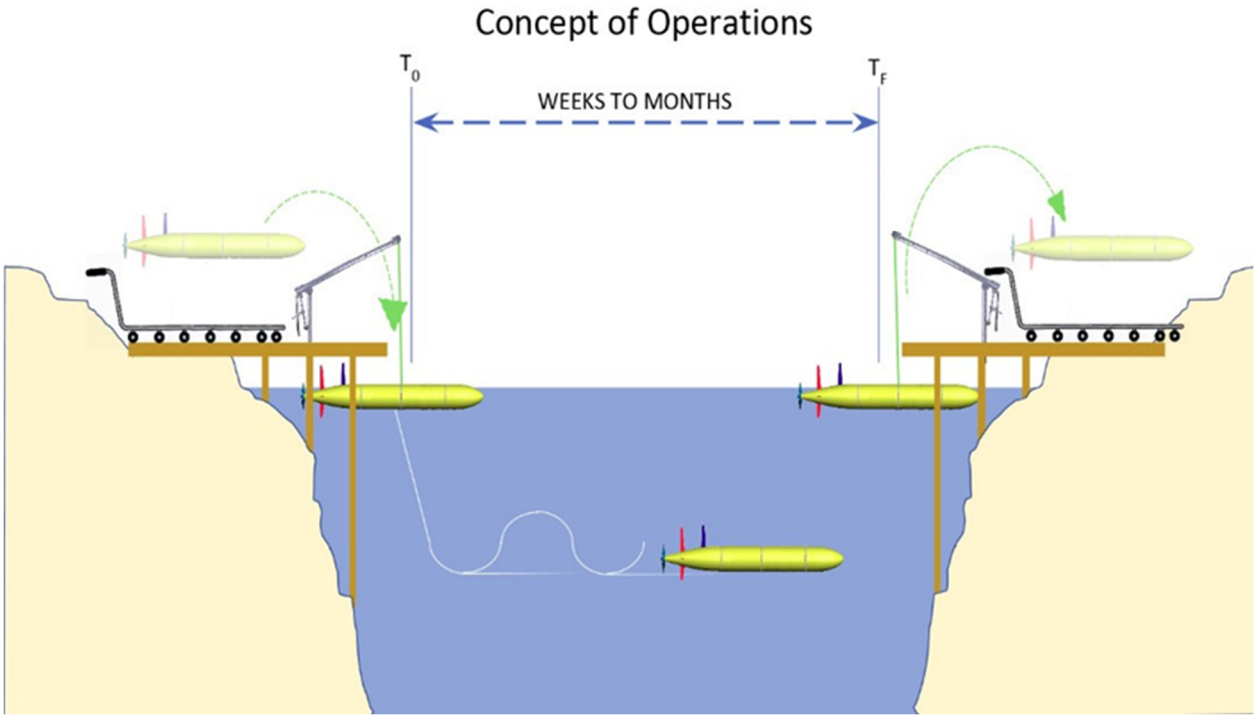


Figure 2-1: *The Huracan Concept of Operations*

Figure 2-2 shows a notional physical configuration of the system. Physically, a 21" diameter hull will be used, and vehicle length, which can potentially be as long as 20', can be modularly adapted depending on payload and stored power requirements for a particular mission. Computing, navigation, propulsion and power management components will be installed in the rear of the vehicle. Payload components and batteries will be installed in the front and midsections of the vehicle, with the potential to trade-off payload equipment with battery modules given the demands of the mission relating to payload functionality and operational life. We anticipate that onboard power can be ~30 kW in its highest power configuration.

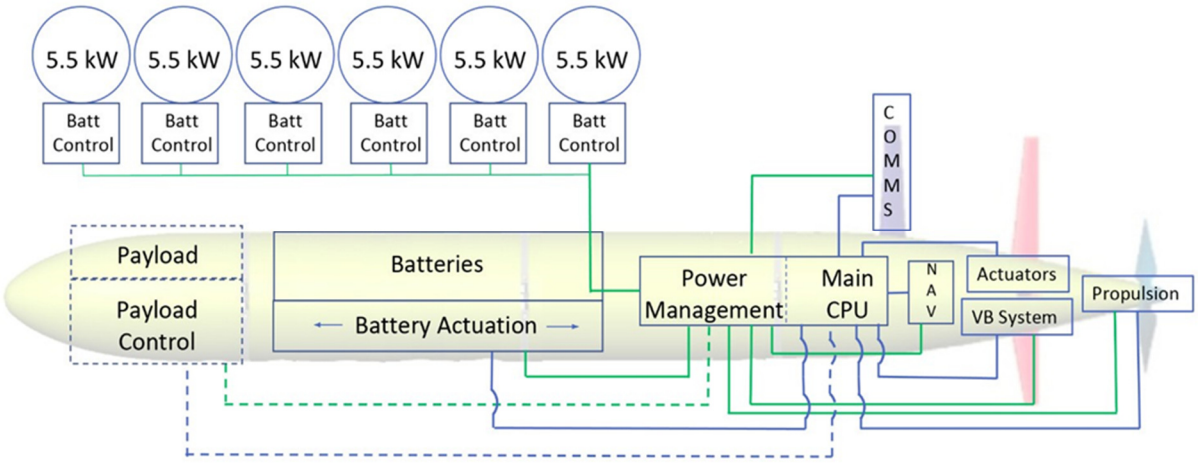


Figure 2-2: *The Huracan Physical Configuration*

In terms of subsystems, the notional system design includes the following:

- The distributed computational system running custom software will consist of subsystem-oriented motherboards connected to local components/peripherals and networked via a CANbus throughout the vehicle, to include between watertight equipment enclosures;
- To support navigation, sensors will include a Doppler Velocimeter, an Inertial Motion Unit, GPS for surface navigation, and sonar for forward-looking and altimeter functions;
- Actuators will include a thruster for propulsion, rudder (and if needed elevator) surfaces, a variable buoyancy system, and a battery positioning system to alter the center of mass, and therefore the nominal glide angle, of the vehicle. This suite of actuators enable standard AUV navigation with modes for power efficient descent/ascent and neutrally-buoyant loiter. Figure 2-3a shows an example of an MBARI-designed weight shift mechanism similar to what we plan to use for pitch control. The Huracan mechanism, however, will be redesigned and installed in the wet section of the hull in order to shift 2 to 3 of the battery spheres using a set of rails and

rollers with a driver motor built to operate in a wet environment. The number of batteries to be shifted has yet to be determined;

- Contrary to our original plan, we are not using an extended version of the MBARI battery system, rather we have designed a power system around the latest available cell module from the same vendor that supports the MBARI design. This new module provides greater energy-density, and a custom interface has been developed to fully exercise its capabilities (beyond that which is normally available directly from the vendor). These cells will be arrayed in parallel and, mimicking the MBARI-approach, will be packaged in 6000 meter rated glass spheres (such as the old MBARI model shown in Figure 2-3b) with SCU-developed custom management electronics for balance, and charge/discharge control.
- The hull design will be based on using the long body sections frequently employed on Bluefin Robotic vehicles, shown in Figure 2-3c. Experience with the Dorado class systems and Bluefin 21" systems have shown we can control two of the extended length sections with a tail and nose cone. Doing this we should be capable of housing 5 or more spheres in a vehicle depending on the need for payload space. We would like to try to incorporate as many as 8 battery spheres for demonstration purposes, but this depends on the affordability and availability of high-performance syntactic foam which will be required, which limits depth.
- Various wireless communication links will be possible to support very close-proximity high data-rate wireless operations and piloting (range of 10s -100s meters) via a wifi or similar connection, a long range (1-100s miles) command and telemetry link via cellular technology (when in range of commercial cell networks), a long range low data-rate beacon service available globally (possibly via a satellite link) for reporting position and health, etc.;
- Emergency Service equipment, such as a ballast drop mechanism, will be included. The ballast drop mechanism, in particular, will be actuated in emergency conditions (such as navigation failures) such that it is desired to have the vehicle surface;

a. Example Mass Slide Actuator for Pitch Control *b. Battery Sphere* *c. Body Sections*

Figure 2-3: *Examples of the Type of Components Planned for the Huracan AUV*

In terms of mission navigation modes, we anticipate the need for the following functions:

- Close range remote joy-stick based control for near dock piloting;
- GPS-based waypoint and/or path-following surface navigation;
- Underwater waypoint mid-water navigation;
- Mow-the-lawn survey-style navigation with sequential depth/altitude planes;
- Dwell mode with bottom-sitting, drift and/or orbit options;
- Adaptive navigation capability to track and follow signals of interest.

Beyond navigation functions, there will also be on-board software to support instrument/payload management mission definition/planning/execution, diagnostics and fault detection/handling, and so on.

3.0 Distributed Computing & Power Control Architecture

This section details the adopted architecture for the low-power distributed computing and power control system. In doing so, it first reviews the decisions behind the selection of the processing approach and ultimately the selection of a specific microcontroller and processor development board adopted for this effort. Second, given these decisions, the general notional concept for the distributed system is summarized.

3.1 Processing Topology/Protocol/Component Trade-offs

Several architectural choices were made based on trade-off analysis given the needs and requirements of anticipated Huracan users/customers/partners. Several of these are briefly reviewed here and are covered in more detail in a related project report.[1] The main trades considered the computing topology, the primary bus communication protocol, the choice of microcontroller vs microprocessor, and the selection of microcontroller family and ultimately specific components.

3.1.A Communications Topology: Linear vs Star. A trade study was executed to understand the benefits of linear bus topology and a star topology. As stated in the original proposal, SCU has considerable experience with the development of linear bus distributed computing topologies, having developed such systems for spacecraft and US Air Force / NASA grants. MBARI's LRAUV uses a star topology.

In a typical linear bus topology such as that shown in Figure 3-1, all computing devices are linked to each other with a common interface via a shared "bus" such that every device can send and receive messages to/from all other devices. Various software-defined protocols may be adopted to coordinate such communication, ranging from a) a "master" mode in which one computer coordinates all communications and the others only speak when spoken to, to b) a "multi-master" protocol where there is no single device that has hardware control over the network. The software basis for protocol implementation promotes flexibility in implementation and maturation. Examples of linear bus approaches include buses using the I2C and CAN protocols/components, with standard bus lines which may include things like data lines, clock lines, etc.

The other topology that was considered was a star topology, shown in Figure 3-2, which is typically implemented using a protocol such as SPI. There are several benefits to this approach, namely a dedicated connection to each device in the network. By having a dedicated line to

[1] Z. Cameron, "Distributed Computing and Power Control Architecture for a Long Range AUV," Capstone Report, MSEM&L, Santa Clara University, Adv: C. Kitts, June 2021.

each device, there is less of a risk of communication collisions or interrupted communication. Additionally, there is a finer control over the devices in the network because there is one main device that controls the network,which is the device at the center of the star. Finally, it is often the result that when a star topology is used, faster data rates can exist between the center device and the other devices in the network. Interestingly, the “dedicated line” main benefit is indeed advantageous if every device is in a similar location; however, if the devices are distributed in different parts of the vehicle, running multiple lines to each part of the system can be quite cumbersome. When a new device needs to be added, a new communication line must be run between it and the center device.

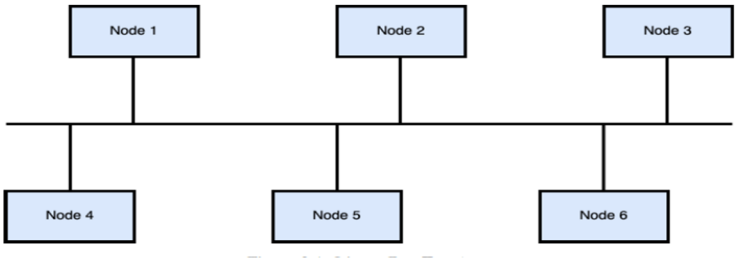


Figure 3-1: *Linear Bus Computing Topology*

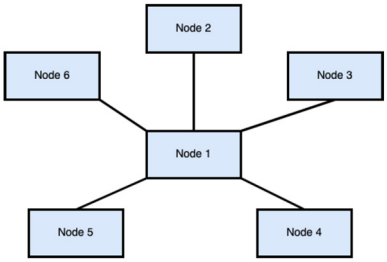


Figure 3-2: *Star Computing Topology*

Given the aforementioned pros and cons of each topology, a linear bus topology was selected. The most significant reasons for this decision was its appropriateness for a system with distributed components (some ~10-20 ft away which would make running new lines particularly cumbersome), the desire for parallel task processing (which is naturally supported in a distributed computing architecture), ease of integration (given the common hardware/software interface), it’s ability to assign proportional computing/power resources to required tasks (only nodes supporting active tasks need be powered), the developmental advantage of not requiring a single software engineer to integrate all device nodes via a central processor, and SCU’s experience with the technology.

3.1.B Communications Standard: I2C vs CAN. Given the aforementioned choice of a linear bus, a follow-on trade study was conducted to evaluate the pros and cons of the two primary linear bus interface candidates: I2C or CAN.

I2C is a two wire serial data bus that is primarily used for connecting multiple devices to one single device. One wire carries a clock signal generated by one device to the rest of the devices on the system. This line is connected to the serial clock (SCL) terminal of all devices on the bus. The other wire is a data wire that carries data from any device on the network to all other devices on the network. This line is connected to the serial data (SDA) terminal of all devices on the bus. Additionally, both lines are connected to pull up resistors to create a common voltage state, as shown in Figure 3-3.

All communications to devices on the bus are addressed, most commonly by a 7 bit address, where the mother device specifies an address of a daughter device that it wants to communicate with. There are several advantages to this protocol. The first advantage is that I2C is a standard protocol on many embedded devices and systems. There are far more devices that support I2C natively than CAN. Additionally, I2C requires no external transceivers, but rather the devices themselves are capable of producing the required voltage levels and logic for the bus to operate. Additionally, another advantage of I2C is that packet length is undefined, meaning that a device could theoretically send as many bits as required in one packet rather than having to send multiple distinct packets. Additionally, because all devices have the SDA and SCl wires go to the same terminals, there is no need for different wiring for various components within the system like there might be on a UART serial protocol.

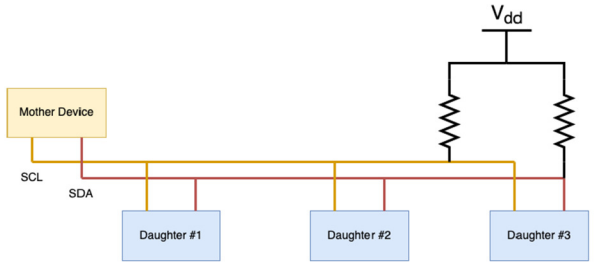


Figure 3-3: *I2C Bus Connections*

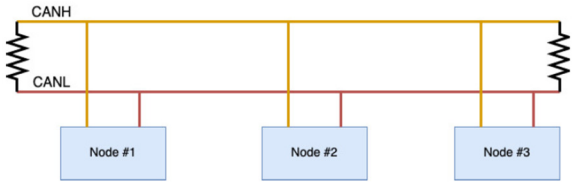


Figure 3-4: *CAN Bus Connections*[2]

There are disadvantages to I2C though. One of the biggest is due to the nature of the communication style. I2C communications are subject to interference and distortion, creating data corruption in longer runs of cabling. While there is no explicit maximum length that the cable can be, there is evidence to believe that the cabling is best left at a few meters maximum. This is generally not an issue for most small robotics applications, but if a cable is to be added that spans an entire vehicle, this will be a consideration as the craft will be greater than seven meters long (and wiring is often significantly longer than the “as the crow flies” distance between devices). In order to increase the length of this I2C run, lower transmission rates and signal repeaters would have to be utilized, which would significantly hamper the performance and power efficiency of the vehicle. Additionally, because the protocol is address-based, there is less ability to do custom addresses and protocols as the address format adds a level of rigidity that is not present in other communication protocols. Finally, I2C has no built-in error checking or collision avoidance within the protocol, so all control and error checking must be done at the application level. This is especially important within a linear bus system. While I2C does not prohibit multiple master nodes driving the bus, there needs to be software that prevents the

[2] ”Introduction to the Controller Area Network (CAN)”, ”Texas Instruments”, 4 2016.

devices from collision which can add a layer of complexity to creating a reliable communication network.

CAN networks differ from I2C networks in that, while the protocol is made up of two lines; these are referred to as a CAN High (CANH) and CAN Low (CANL) with a typical layout shown in Figure 3-4. Rather than a data and clock line like those found in an I2C communication protocol, CAN networks use a differential signal to send a packet. This relies on a dominant state and a recessive state where the line represents a zero and one in those respective states. Figure 3-5 shows an example of this. In a recessive state, both the voltage of CANH and CANL are held at 2.5V. When a transmission occurs, a zero is created by driving the voltages apart, with CANH going to the higher voltage and CANL goes to the lower voltage. The voltages will continue to separate like this for the entirety of the transmission until the packet is finished transmitting.

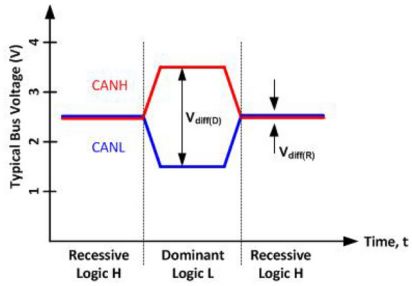


Figure 3-5: *CAN Voltage Levels*

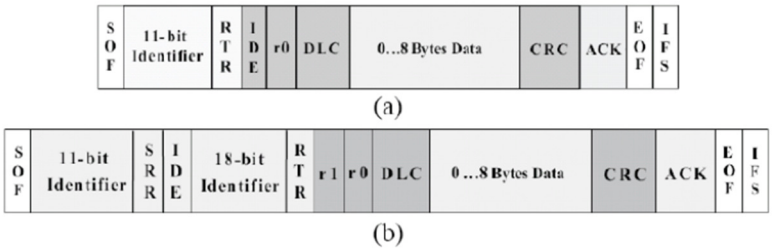


Figure 3-6: *CAN Packet For Versions 2.0A/B*[3]

For wiring the CAN lines together, the system relies on a twisted pair of wires. On each end, there is a 120 ohm resistor that connects CANH and CANL. This use of twisted pair and differential signals allow for the signal to be protected by the electric fields the signals create. The opposing electric fields help to reduce noise and distortion of the signal as it travels through the bus. It is this property of differential signals that has a unique advantage over other forms of serial communication. This allows CAN signals to be transmitted over longer distances with a maximum distance of 40 meters even when transmitting at its maximum speed of one megabit per second. Small runs of cabling can be done without the assistance of a twisted pair, but the travel distance is far smaller as a result. Another key characteristic of CAN is the way in which the devices are connected to the bus. The messages and the network are not based on addresses of devices, but are instead based on messages and their content. A message-based communication system highlighted a potential ability to have both a client-server communication system as well as a producer-consumer based system.

[3] W. Lun, C. K. Ng, B. Mohd Ali, N. Ali, F. Noordin, and F. Rokhani, "Review of researches in controller area networks evolution and applications," Proceedings of the Asia-Pacific Advanced Network, vol. 30, 08 2010.

There are some drawbacks to CAN, and the biggest is power consumption. Many devices, even if they can transmit and receive CAN digital data, do not have a built-in transceiver to convert those digital signals to CANH and CANL signals. To overcome this, additional CAN transceivers have to be added to each device on the bus, increasing the overall current draw of the system were a CAN bus to be employed. CAN transceivers can have some power-saving functionalities, however a CAN transceiver that was tested in this process, the SN65HVD230 by Texas Instruments, was commonly seen consuming up to 10mA even in idle. While this power consumption can be reduced, it is certainly a consideration when trying to lower consumption as much as possible. Additionally, as mentioned before, CAN is not as common of a protocol on microcontrollers as I2C or SPI. The choice of microcontrollers would be limited to devices that supported CAN, which was another consideration. Finally, CAN 2.0 only allows a maximum of 8 bytes of data, however that data packet can be identified with 11 bits in the CAN 2.0A standard and 29 bits in the CAN 2.0B standard, which allows the data fields to carry actual data. The layouts for both the CAN 2.0A and B standards are shown in Figure 3-6.

CAN was ultimately decided as the communication protocol of choice for the development of the prototype. Because of its strong mixture of strengths over an I2C bus, especially when it came to collision avoidance and built in error checking, CAN was seen as the best option to pursue. Also, CAN has flexibility in communications protocol and can support longer cable runs. Finally, interest was expressed by the MBARI team that CAN was something they have used in the past and had positive experiences with. While power consumption is still an issue that is being monitored by the team, without proper knowledge of an exact power budget, it has been hard to quantify the impact that power consumption of a CAN network will cause.

3.1.C Computing: Microprocessor vs Microcontroller Nodes. Most processors identify as being part of two main groups: microcontrollers and microprocessors. With these distinctions, there is generally an ideal use case for each kind of processor. Accordingly, the benefits of each form of processor were weighed and evaluated.

Microprocessors, often called application-level processors, are processors designed for computation as one of its primary characteristics and marketable traits. Microprocessors can range in power consumption and computational proficiency, but the suitable choices for this project were single-board computers. Microprocessors on such boards typically are capable of running general-purpose operating systems (GPOS), such as a Linux image. Additionally, current microprocessors, even those found on single board computers, can have multiple cores to allow for parallelization of tasks. With these factors in mind, it is fair to characterize that a microprocessor's goal is the maximization of throughput. Throughput is a necessity when dealing with complex tasks and many tasks at once, however it is not normally a necessity for control systems.

Microcontrollers are the second form of processor that was considered. Processors in this range are generally used for their low cost and focus on control of peripherals. Microcontrollers can range vastly in terms of their capabilities, such as in common 8 bit microcontrollers like the one found on the popular Arduino Uno to 32 bit processors capable of multiple different communication protocols. While most microcontrollers are incapable of the performance afforded by a generic microprocessor, they are generally capable of doing small computations as well as providing sufficient resources for the control of multiple peripherals. Microcontrollers have continued to evolve, such as adding floating point units to increase computational performance. However, this increased throughput over previous years is a bonus rather than the main purpose of microcontrollers. Microcontrollers, while mostly incapable of running a GPOS, are capable of running real-time operating systems (RTOS), which focus on creating deterministic systems and reducing latency. Microcontrollers are often more appealing to use with an RTOS compared to microprocessors because there is a concerted effort to reduce latency for microcontrollers. Even if a RTOS is not employed with a system, microcontrollers retain their competitive advantage of lower input latency in the same way that microprocessors have a competitive advantage in throughput. One additional point to consider in the topic of power consumption. Due to their lower computational power and lower clock speeds, microcontrollers as a whole are known to have a lower power consumption than a microprocessor. Additionally, microcontrollers will often have different power modes, where the processor can shut down a section of the chip or clock to lower speeds in order to consume less power during different modes of operation. Microprocessors often do not have these predetermined hardware power states and instead rely on software to monitor their efficiency.

Microcontrollers were chosen for this project. Microprocessors, while generally faster and more computationally advanced than microcontrollers, often consume more power, have higher costs, and are more focused on throughput than latency. Meanwhile, many microcontrollers are capable of doing the control and calculations necessary of the gateway units within the distributed architecture we have adopted while providing lower power consumption and better functionality within a real time system. As a result, the search for a processor within the gateways was narrowed to microcontrollers.

3.1.D Choice of the ARM Cortex-M. Microcontrollers are developed by a variety of different groups. They range in capability between small processors capable of performing instructions using 8 bit math, to larger processors capable of performing 32-bit computation. Due to this diversity of options, it was important to analyze trends within the industry to evaluate the correct kind of processor to select. Eventually, it was decided that an ARM processor would be used as the foundation of the gateway. This decision was largely based on the diversity of offerings that ARM has, the relationship it has with other companies who create products

around their specifications and instruction sets, as well as the market share and amount of available products to choose from.

Additionally, ARM processors have a variety of devices that range in varying levels of power consumption and performance. Their Cortex-M line focuses on creating microcontrollers with varying levels of capabilities. In total, there were three main groups of ARM Cortex-Ms that were discussed: the Cortex-M0+, the Cortex-M4F, and the Cortex-M7. To conduct a general evaluation, a trade study was opened by looking at the Teensy product line, a group of hobbyist microcontrollers that showcase the characteristics of each class of processor, such as available connections and generalized power draw comparisons between the processors:[4]

- Cortex-M0+: In comparison to other microcontrollers in ARM's product line, the Cortex-M0+ is not the most impressive processor. However, it is fully capable of utilizing the Thumb-2 instruction set that the other processors on this list are capable of using, even if the processor is slightly slower compared to the other options. Cortex-M0+ often have lower power consumption than microcontrollers in the other classes. One factor to consider is their connectivity options are fairly low, with many chips not having a CAN input and output like the other options do.
- Cortex-M4F: The Cortex-M4F is a solid middle ground between the two other offerings. Not only are there processors in this class that offer low power consumption in line with the offerings in the Cortex-M0+ range, but they also provide a solid mix of connectivity. This is the first range where a CAN TX and RX can be commonly found, along with more general purpose interfaces like UART, SPI, and I2C which were present in the Cortex-M0+ offerings. Additionally, the Cortex-M4F has the option of higher clock speeds as well as a built-in floating-point unit to have hardware-accelerated floating point calculations, vastly improving performance over the Cortex-M0+.
- Cortex-M7: One of the most high performing microcontrollers in the ARM product line is the Cortex-M7. This processor utilizes the same instruction set as the previous models, but has vastly higher performance. With most clock speeds several hundred megahertz higher than the Cortex-M4F, it is much more computationally proficient. This results in a higher power consumption and therefore should be used in scenarios with the proper consideration.

To evaluate the power efficiency of the ARM options, the Teensy line of microcontrollers was picked to compare the processors. The Teensy line provides multiple options utilizing ARM Cortex-M processors, making it easier to compare the different styles of chips to one another in

[4] "Teensy® usb development board." [Online]. Available: https://www.pjrc.com/teensy/

a real world scenario. The boards selected were the Teensy LC, the Teensy 3.6, and the Teensy 4.0 to examine the Cortex-M0+, Cortex-M4F, and the Cortex-M7 respectively.

To examine the low power states of the boards, the Teensy Snooze Library was used, an open source Arduino library designed specifically for the Teensy line to test their power consumption at various states.[5] The results of the various power states are tabulated in Table 3-A. As expected, the Cortex-M0+ board had the lowest power consumption with the Cortex-M4F lagging slightly behind. The least efficient board was the Cortex-M7 based offering, with significantly higher power consumption in all modes of operation. This higher power consumption made it increasingly more difficult to justify the use of a Cortex-M7 as the power consumption of the chip would prevent a suitable low power system. Additionally, because CAN was decided as an option for the central communication between the different subsystems, the justification for the Cortex-M0+ was difficult as it provided less I/O options and did not include a native CAN connection. As a result, the Cortex-M4F seemed to be the most balanced offering from ARM and was used as the processor of choice when evaluating different options.

Table 3-A: *Power Consumption for Various Teensy-Line Microcontrollers*

	Teensy LC	Teensy 3.6	Teensy 4.0
Idle Current (mA)	9.95	84.8	106.6
Idle Watts (mW)	49.75	424	533
Sleep (mA)	0.553	18.31	51
Sleep Watts (mW)	2.765	91.55	255
Deep Sleep Current (uA)	211	308	8925
Deep Sleep Watts (mW)	1.055	1.54	44.625
Hibernate Current (uA)	4.5	82.4	6703
Hibernate Watts (uW)	22.5	412	33515

3.1.E Choice of the STM32L432KC. Once the Cortex-M4F was selected as the best choice, a specific processor needed to be found that had a good balance of power management, peripheral support, and compute capabilities. The search for a processor was done by analyzing the offerings of multiple companies. ST was chosen as the company to select the processor from because of their availability of free software applications to program their chips, as well as a diverse product line with offerings specifically targeted for low power operation. The search was narrowed to devices that had support for CAN, multiple power management states, and

[5] duff2013, “duff2013/snooze,” Oct 2020. [Online]. Available: https://github.com/duff2013/Snooze

ability to buy on a development board with the desired chip for evaluation and prototyping. The microcontroller selected was the STM32L432KC, which is pictured in Figure 3-7.[6]

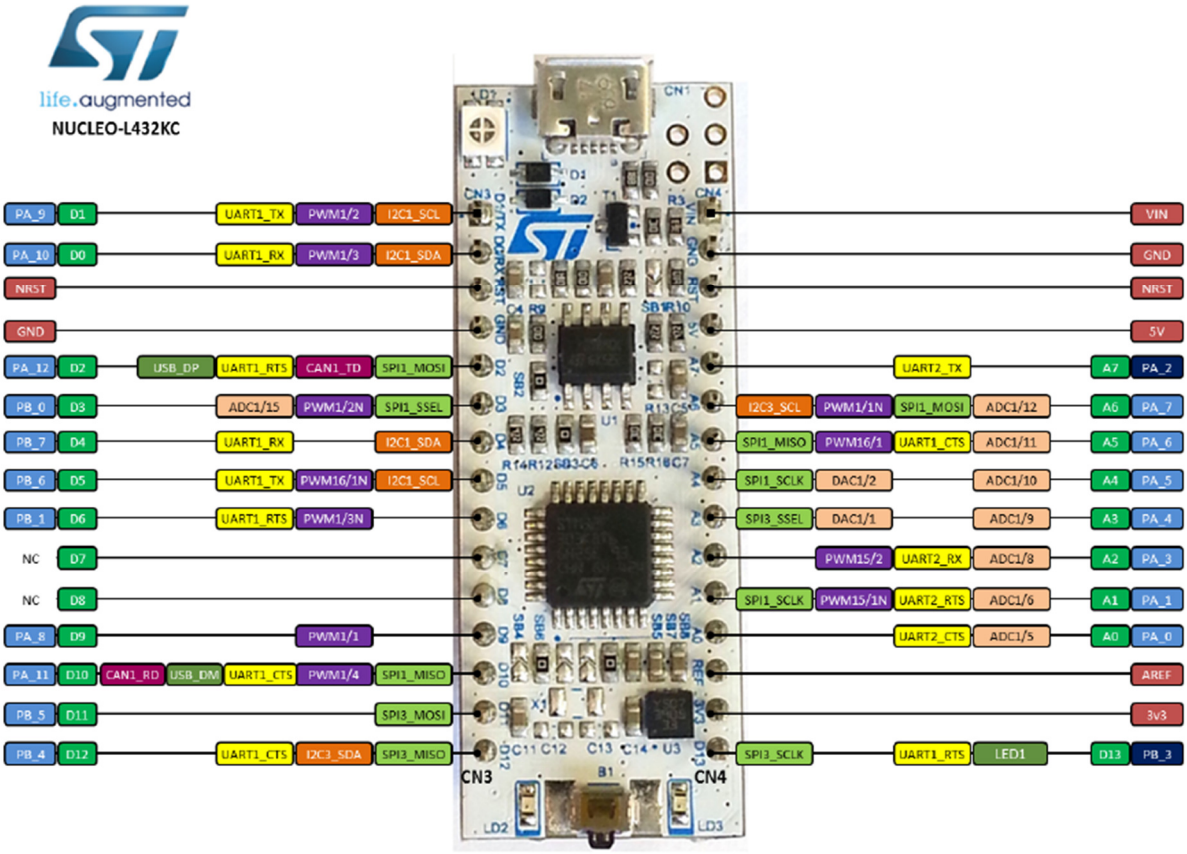


Figure 3-7: *The NUCLEO-L432KC Development Board, showing functionality of each pin*[7]

One of the biggest drivers of this selection was the chip's power consumption, as it provided multiple low power states, such as standby and shutdown, which can be utilized for different situations and reduce power to varying degrees. The data sheet claims power consumption of 8 nA in full shutdown mode which could be ideal to reduce power consumption in the future. Additionally, it has access to a CAN transmit and receive, which allows for a CAN transceiver to be connected directly to the processor. This goes along with other capabilities like UART, SPI, I2C, ADC and PWM. Finally, being a Cortex-M4F, it has access to a floating point unit that makes

[6] "Ultra-low-power Arm® Cortex®-M4 32-bit MCU+FPU, 100DMIPS, up to 256KB Flash, 64KB SRAM, USB FS, analog, audio", "ST".
[7] "Nucleo-l432kc." [Online]. Available: https://os.mbed.com/platforms/ST-Nucleo-L432KC/

it capable of doing floating point math without software emulation. Another consideration was that ST offers free software tools to develop on their platform ranging from code generation tools, interactive development environments (IDEs), debugging tools, and programming utilities. This suite of tools and the functionality of the chip made it possible to prototype the system and showcase the capabilities of a distributed system. Development boards with the chip are available as well, namely the NUCLEO-L432KC. This allowed for the rapid prototype and development of the different systems that the architecture features.

3.2 Distributed Computing System Concept

Given the choices reviewed in the previous subsection, the team adopted a linear distributed computing architecture using a CAN bus as the primary inter-node communication service. As such, the notional architecture is as shown in Figure 3-8.

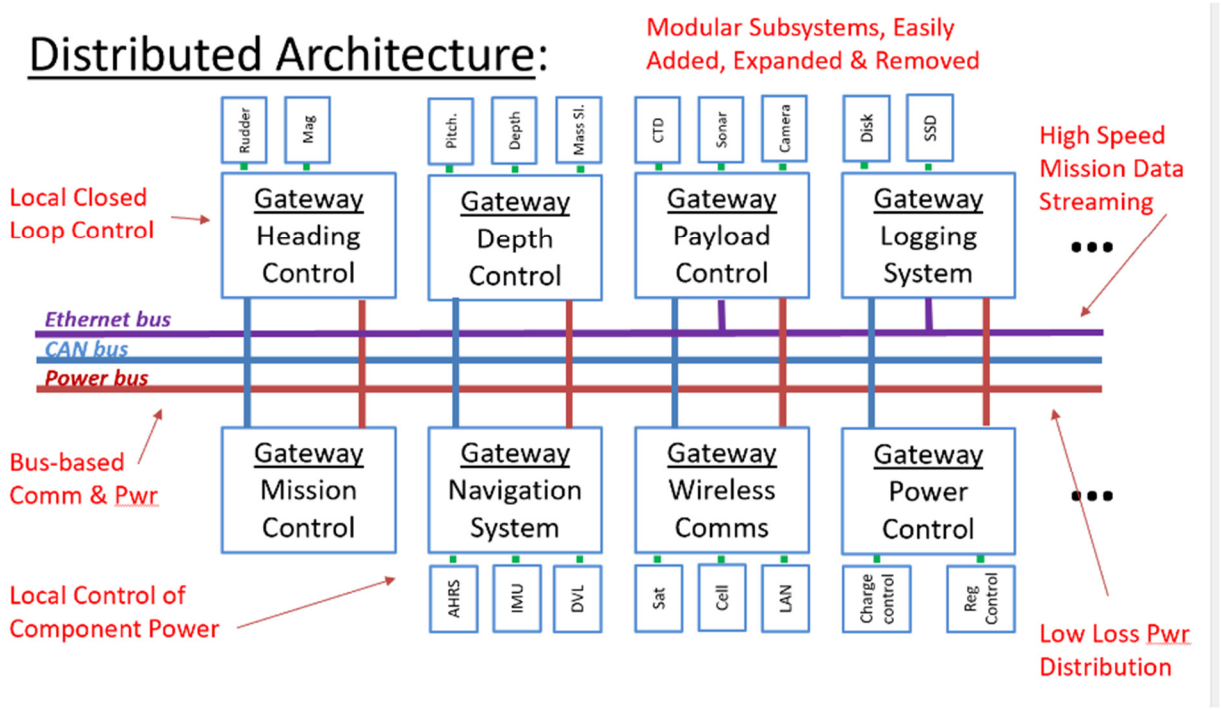


Figure 3-8: *Distributed Computing and Power Control Concept Architecture*[8]

The architecture consists of a number of computational 'gateways' that perform local power configuration and functional control capability. These gateway nodes are connected via a CAN bus to support communications across the entire system. The CAN selection dictates the nature

[8] C. Kitts, Slide as part of "MBARI-Santa Clara University Partnership in Marine Technology Development and Education," Presented to Navy Admiral, MBARI, March 2022.

of the physical and data interface, but there is flexibility in the manner in which communications are coordinated between gateway nodes.

While this overall approach is compatible with any computational node that “speaks” CAN, the SCU team developed a new custom low-power gateway with many of the common features deemed necessary for the Huracan system. That said, we have also demonstrated the ability to deploy other computational nodes in the same system, such as a Raspberry Pi. This allows a wonderful level of flexibility, allowing most of the system to be controlled through the customer gateways but to deploy, as necessary, more capable processors, such as Pi’s, Jetson boards, and so on.

As seen in the Figure, the gateways have functional task-oriented assignments, such as payload control, heading control, mission control, etc. As such, the gateways are connected to appropriate components/peripherals to support those tasks. The gateways are low-power boards to begin with, but they also have the ability to turn power on/off to individual components (and their interface circuitry) depending on the state of task processing. The gateways allow an external controller (automated or manual) to dictate operation of its components, and they also are able to accept higher level commands and to operate their components accordingly without the need for additional external guidance (or power use). For example, the heading controller can accept low-level device commands for the rudder and compass, but it can also accept a higher-level heading setpoint command and execute a realtime PID control loop that uses the capabilities of those two components. The modular system allows new functional gateways to be added or removed without impacting the implementation of other subsystems.

The Figure also shows a power bus that provides high voltage power across the entire system via a single line. This line is sourced by the battery system, which is managed by one of the gateways. Voltage conversion to lower voltages required by components happens locally, providing a low-energy-loss approach to power distribution.

The Figure also shows an Ethernet bus. This is used only for high bandwidth routing to a mass storage device to accommodate payload data (or other components producing such data). This is provided for mass data storage not required for realtime vehicle operation. The separate bus segments this data flow from the CAN bus. As a result, this high bandwidth flow of data, which can lead to packet loss and communications latency, does not negatively impact platform monitoring and control operations being implemented via the CAN bus.

4.0 Description of Distributed Computing Node Motherboard

This section details the design of the custom computational gateway boards developed for this program. These boards are a critical element of the low-power design given their use of modern low power components, their power management features, the high-resolution power control capability, and their ability to achieve local functional real time control without the need for external processing.

4.1 Introduction

The design goal of this project was to create a distributed system, where at its core, a generic subsystem was used for different purposes. We call this generic subsystem node the 'Ideal Gateway'. As explained in the earlier sections, this gateway is a low-power distributed system which drives the UAV. The low power ARM Cortex-M4 forms the basis of the board. The board is designed to interact with various devices supporting modern as well as legacy protocols, for example: I2C, SPI, CAN, RS232, RS485, SAI, UART, etc. To drive an UAV, devices such as sensors (GPS, depth, IMU, etc.), actuators (thruster, rudder, mass slider, etc.), payload instruments, etc. form the core functionality of the craft. Additional functionalities include data logging, wireless communications, emergency ballast systems, and so on.

From a process perspective, the team started with a schematic design and then progressed to an initial protoboard implementation. Iterations were performed in order to establish proper functionality and to test the circuit in a limited manner with the CAN bus, components, and so on. The team then progressed to designing a printed circuit board (PCB) so as to replicate the design in a neat and compact solution. The design of this board is such that the development team is able to use the board to add various other peripheral devices that were not taken into consideration when first planning to build the circuit. We make a note that this design approach made the distributed board a very modular circuit in which it is up to the team to decide what are the components that are to be populated. This helps in making the Ideal Gateway a common generic distributed node used for various types of functions.

4.2 Design

The computational capability of the gateway comes from the ARM Cortex-M4 board. Each gateway shares a common set of software for functions such as CAN communications, peripheral interface support, etc. Furthermore, each gateway can be programmed with node-specific software given the tasks it is to manage, the components it interfaces with, etc.

The node is divided into 8 different parts including the M4:

1. ARM Cortex-M4
2. GPIO Expander
3. CAN Module
4. I2C bus
5. SPI bus for data logging
6. High power power switching
7. UART Bridge - RS485 and RS232
8. Power lines

Figure 4-1 shows how the different modules are connected to each other. The components used for the modules which are not in use not only have a low quiescent current value but are also cut off from the power supply using a combination of FETs signaled by the GPIO Expander. The UART Bridge has modules to enable legacy communication protocols such as RS-485 and RS-232. An important application of the Ideal Gateway is to power switch external components. The power switching conserves energy by only allowing power to the external device when it is needed. The GPIO Expander is used to trigger the Dual Latch Power Switching relays. Another important aspect of the board is the distributed capability enabled by the CAN messaging system. A CAN module integrated with the board interacts with the M4 to relay the messages to and from the CAN bus.

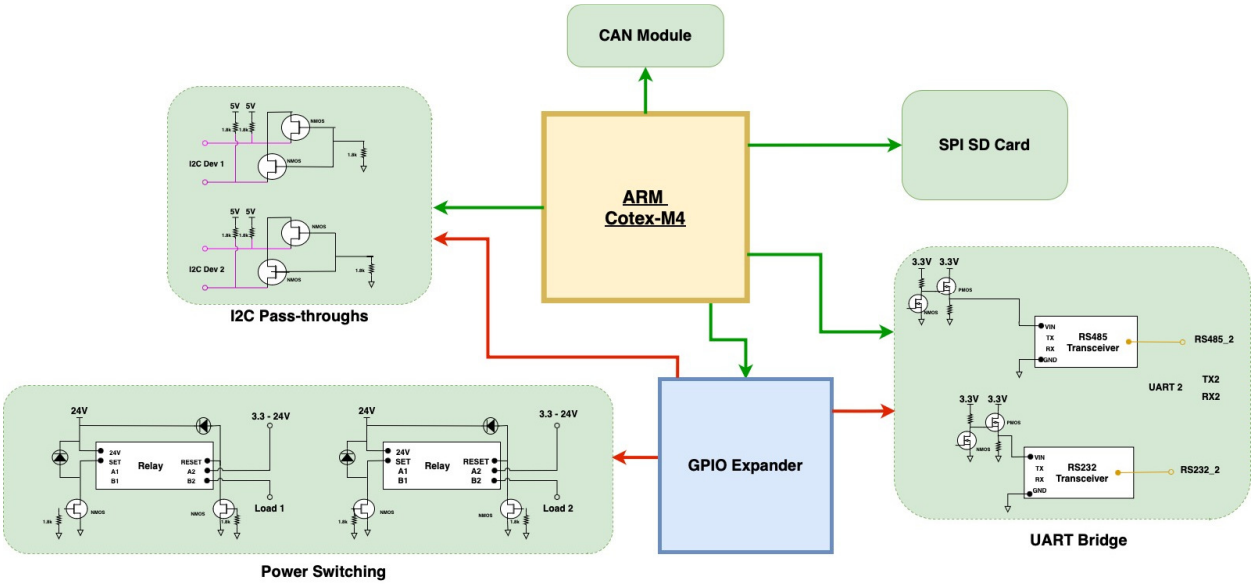


Figure 4-1: *Block diagram of distributed node called the 'Ideal Gateway'*

Each of the modules communicate with the Cortex-M4 using different protocols. Below is the list of those communication protocols:

1. UART Bridge - UART
2. I2C pass-throughs - I2C
3. GPIO Expander - I2C
4. CAN module - CAN
5. SPI SD Card - SPI

Over the course of this project, two versions of the PCB design were implemented. Both are based on the block diagram shown in Figure 4-1. The PCBs are designed to be developmental boards on which the team could integrate more devices and perform various other tasks required for the mission. This capability is enabled by the modular design of the PCBs. When designing the board, various changes were made to the schematic of the prototype board to reduce the power consumption. For example, one such change was to introduce a three header male pin to be used to direct power to one of the two modes for the RS232 or RS485 modules. A shorting block as shown in Figure 4.2 was used to make the connection with them to power. To connect either module to power a shunt will be used to switch power since only one of the modules will be used at a time. This variation works on top of the GPIO power cutoff explained earlier in this section.

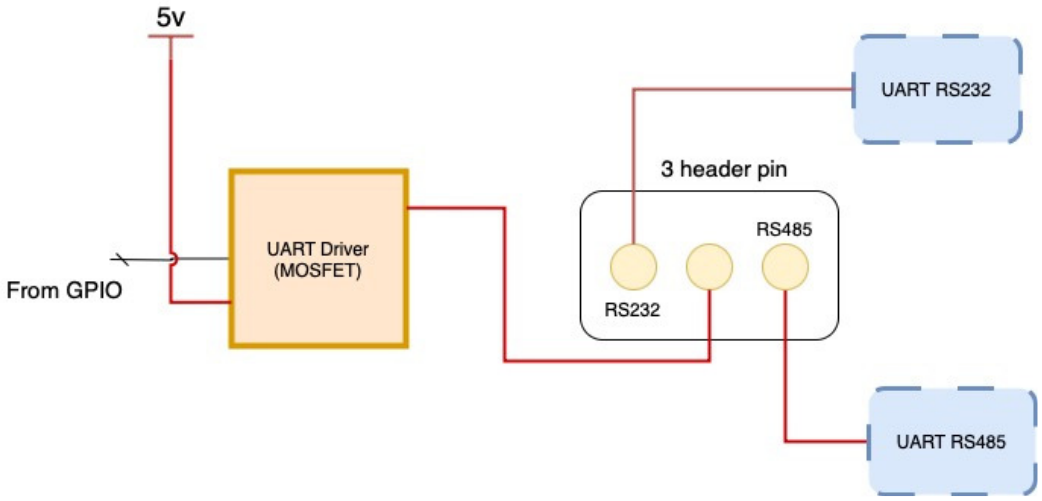


Figure 4-2: Power diverter for RS232/RS485 using 3 pin male header

The software used for the purpose of schematic capture and designing the layout was Autodesk Fusion 360. The software had a learning curve to it: use of the Autodesk Fusion 360 environment, employing cloud based team development, and integrating Solidworks-based CAD design features for electronics casing and harness design. The software has great potential for providing combined solutions for CAD, CAM and CAE design problems.

4.3 Ideal Gateway v1

This section reviews the design aspects of the first PCB made for the Ideal Gateway after the prototype build.

4.3.A. Methodology. As the given prototype had many changes with respect to which pins were to be used for what kind of communication with the other devices and components, the schematic capture part of this version included taking reference from the existing circuit diagram and also making continuous changes to the circuit depending on the requirements of the team. The primary concern for this iteration was to perform as many functions and device integrations as possible on a single distributed node (Ideal Gateway). This would in turn make the gateway more modular and lower the probability of only one pinout handling two or more different functions/communication protocols.

4.3.B Design. The board size for the version 1 is 3.7 x 5.5 inches. We went with a two layer rigid PCB which is optimal for our Ideal Gateway component layout. Figure 4-3 features the layout of the board with the components and traces connecting it.

The board is divided into three main sections:

1. Core and GPIO Expander: This section of the board includes the computational capability of the board to support the different standards used by different peripheral devices connected to the board.
2. Power Switching: In order to show power switching capabilities for high-power signals, one important factor that needs to be considered is the trace width of the track carrying that current. We went ahead with 4 oz/sq.feet copper thickness on the PCB. This copper thickness allows for a greater current carrying capacity. The general formula used for calculating the current carrying capacity of a conductor is provided in Equation 4.1:

$$I = KA/L;l \qquad (4.1)$$

where K is a material dependent constant, A is the area of the cross-section of the conductor and L is the length of the conductor. Hence, to support capacity for 17 Amps of current on the output terminals of the relays, a trace width of more than 196 mil is needed. In order to accomplish this, each side of the board has a 100 mil width track connecting the terminals to the relays.

3. Communication: The communication section is spread out on the edges of the board to make it easier to connect them to different devices. Ports for I2C, CAN bus, UART bridge including two ports for each of RS485 and RS232 and SPI bus are included in this section.

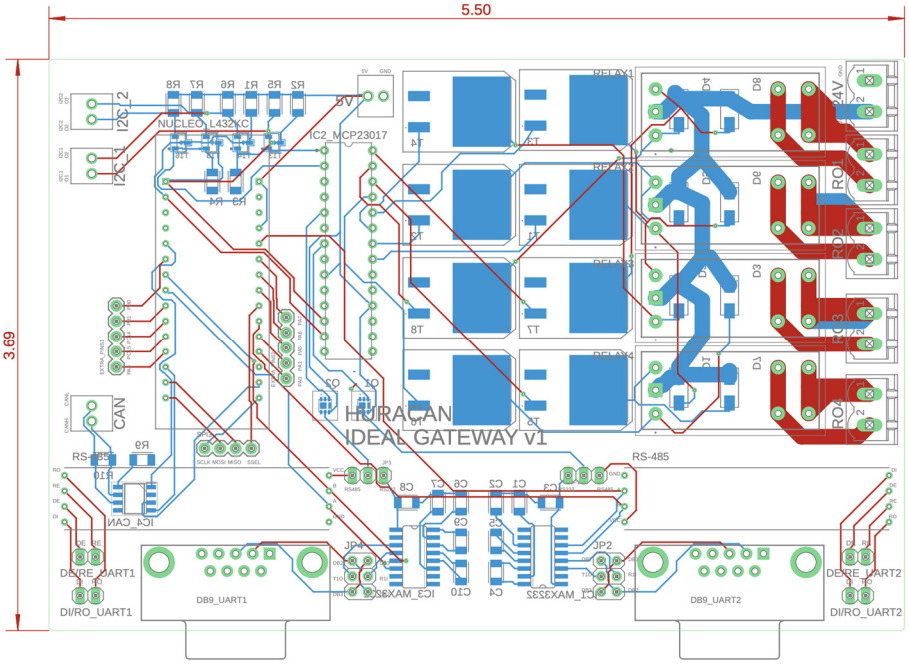


Figure 4-3: *PCB Layout of Ideal Gateway v1*

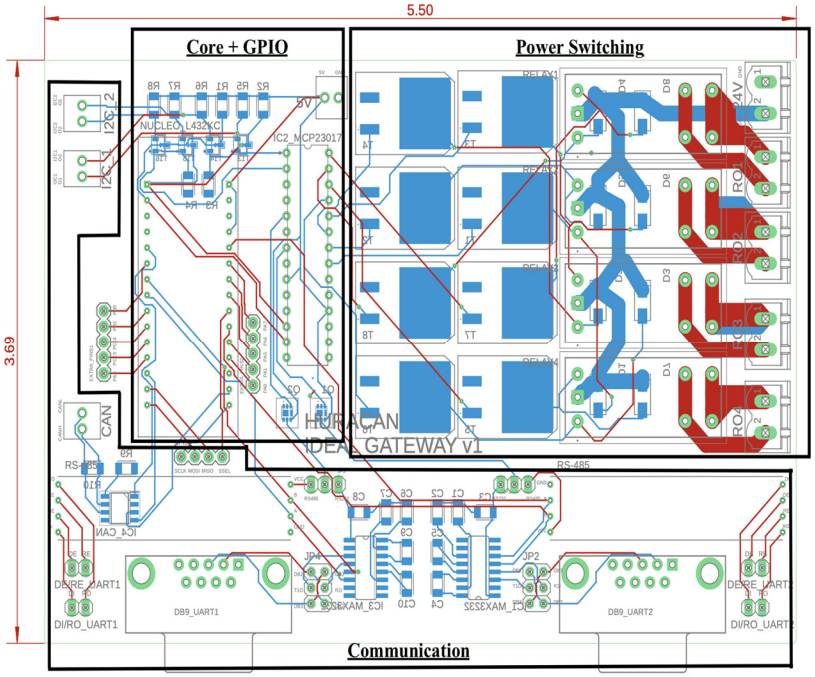


Figure 4-4: *Sectional PCB layout diagram of Ideal Gateway v1*

4.3.C Results. Functional testing of the board was done before moving on to the component level testing. Identified issues included the following:

- Relay power switching: The relays were getting damaged after powering the board and trying to switch them. Further debugging was undertaken to find the issue at its core. The debugging results and the remedy is discussed in the remedies section.
- RS485 pin requirement: The RS485 connection needed the enable pins to be controlled by the Cortex-M4. These pins were connected as pinouts on the board which increased the number of overhead wires from the top. This made the UART RS485 unable to be used on the board with other devices.
- SD Card module integration: While an SPI connection was provided for SD Card support, it became clear that providing a mount for an SD Card was necessary given the importance of providing data logging capability for every gateway.
- Microcontroller flashing: There was a small annoyance in that to reflash the microcontroller, the STM32 board had to be removed either from the PCB or the 5V power to avoid backfeeding power.
- Board legends: The team also encountered the need of proper legends on the PCB so as to connect the components in the right orientation.
- Additional issues: Additional issues included need for surge protection, the need for wider traces to lower impedance for signal integrity, the desire for more breakout and power output pins, the need to integrate breakout boards on the PCB itself, and a more compact design so as to fit inside the UAV designed to test this subsystem.

4.3.D Remedies. While the previous subsection lists many of the issues to be addressed in a new version of the PCB, there was a need to utilize the first version for testing. The primary modification to the board to enable this dealt with power switching. Specifically, MOSFETS were used to trigger the dual latch relays. The gates of these MOSFETS were not pulled down which made the transistor remain in the floating condition due to very small current induced on the gates of the MOSFETs. To tackle this problem, the gates of the MOSFETS were pulled down to forbid the MOSFETS from over-triggering the relays. After this solution, power switching worked as expected.

4.3.E Conclusion. This first PCB revision demonstrated the working principle of the distributed node given that it provided task-level functionality, was able to communicate via the CAN bus, could implement power switching, and was able to be interfaced to external components through a variety of interface protocols. The one functional discrepancy was that RS485 communications could not be tested as the enable pins for the same could not be connected.

This and the above mentioned issues in the Results section are solved in the next iteration of the Ideal Gateway PCB.

4.4 Ideal Gateway v2

4.4.A Methodology. This version of the PCB required changes in the schematic capture to correct the connections for RS-485 modules as well as to integrate more breakout boards such as the RS485 and RS232 modules onto the PCB board itself. Following the reviews from the development team after the testing and use of Ideal Gateway v1, certain design requirements were set in order to make it easier for the team to use the board. The board had to be made compact for which different package sizes were found and integrated in the design. A standard operating procedure (SOP) was made to make design changes to the board which enabled better documentation, quick turnout for producing manufacturing files and less confusion when performing the design change to the schematic and board layout. Continuous feedback from the team using the version 1 helped the design and development of the version 2 Ideal Gateway.

4.4.B Design. The board size was reduced from 3.69 inches x 5.5 inches to 3.75 inches x 4 inches. Major reasons for this compact design were the size of the transistors switching the dual latching relays. The package variant for the transistor used in version 1 was for that of a power transistor. Relay switching doesn't require power transistors to switch themselves. At the same time, pull-down resistors were added to the gates of these switching MOSFETs.

One of the feedback requests from version 1 was to include breakout pins for the Cortex-M4 and GPIO expander as it helps with testing new devices and makes it easier to troubleshoot. Another such developmental design change was to add as many power ports as possible in order to power these additional devices connected to the Ideal Gateway or present in the sub-system.

Another sought after design change was to replace the header pins with terminal blocks which enables us to make proper connection with other devices and nodes. This change will improve the quality of connection from the header pins used in version 1 and decrease the number of instances of bad contact with the conductor.

In order to facilitate the SD card module on every board, a female header pinout to accommodate a stock option SD card module was added to the board. A breakout board option for an SD card is used in order to replace it with another device using SPI communication protocol.

A new 16 pin package for the RS232 chip was found to enable two RS232 ports at the same time. RS485 was also integrated along with the PCB which reduced the space consumption by its breakout board counterpart. The issue of the enable pins for RS485 has been solved in this iteration by providing it the enable, read/write signals from the GPIO/Core combination. The number of ports available on the board for RS485 is one and RS232 is two.

As an example, an IMU sensor using the RS232 standard was integrated with the Ideal Gateway. But, the connector used by this sensor was a 9-pin DB9 connector even though RS232 is based on UART which uses two signals only. Hence there is no need for a 9 connector terminal. Given the aim to make the Ideal Gateway modular, the DB9 connector can be replaced with a normal 2 pin terminal block.

There was sufficient space to widen the widths of certain traces in the layout so as to lower the track impedance. Impedance makes the signal weak and is better when it is a low value.

All the above design specifications and changes done after the version 1 can be noted in Figure 4-5 and the 3D view for the same can be seen in Figures 4-6 and 4-7. The 3D visualization was deemed very useful for considering spatial integration of the boards into the test AUV. This was enabled by Including the 3D library for every component in Autodesk Fusion 360. Finally, the final implemented board is shown in Figure 4-8.

4.4.C Results. The testing of this board will be performed once they arrive from the manufacturers (they have been shipped as of the date of this report).

4.4.D Conclusion. The design changes and enhancements made through the process of designing a second version of the PCB board will certainly improve the utility of the computing gateway. No doubt, new issues will be identified when these boards are put to use. Given the design documentation and use of mature tools, updating the design for subsequent revisions should be a straightforward process.

4.5 Future Prospects

The next steps in the development of the distributed Ideal Gateway node is to integrate the rest of the devices on the board itself. These components include the ARM Cortex-M4 module, SD card module and using SMD packages for components that are currently through-hole. An estimated 20% decrement in the size of the board is expected by doing this. Maturing the board to this level will require improved availability of electronic components.

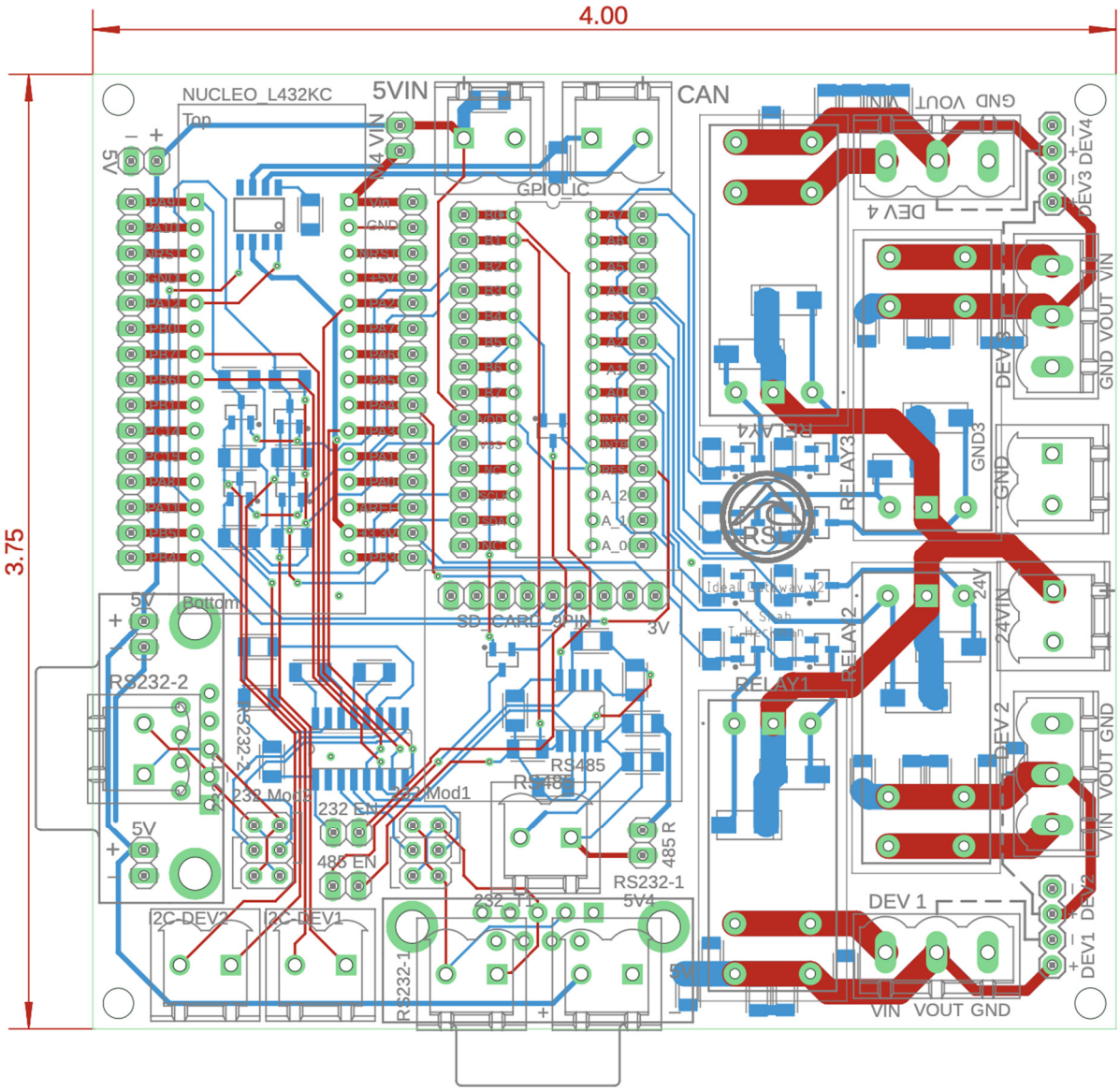


Figure 4-5: *PCB Layout of Ideal Gateway v2*

Multiple signals of different frequency and voltage levels are moving on the board. An important aspect of developing the board further would be to give importance to signal integrity of certain high frequency and low voltage signals. There are many testing simulations provided by the software to do so. Designing the board while maintaining signal integrity principles would enable the team to increase the speed/bandwidth of various standards used on the board such as SPI and I2C. Performing signal integrity tests would give us detailed information about the inductance of certain signals on other signals on the board.

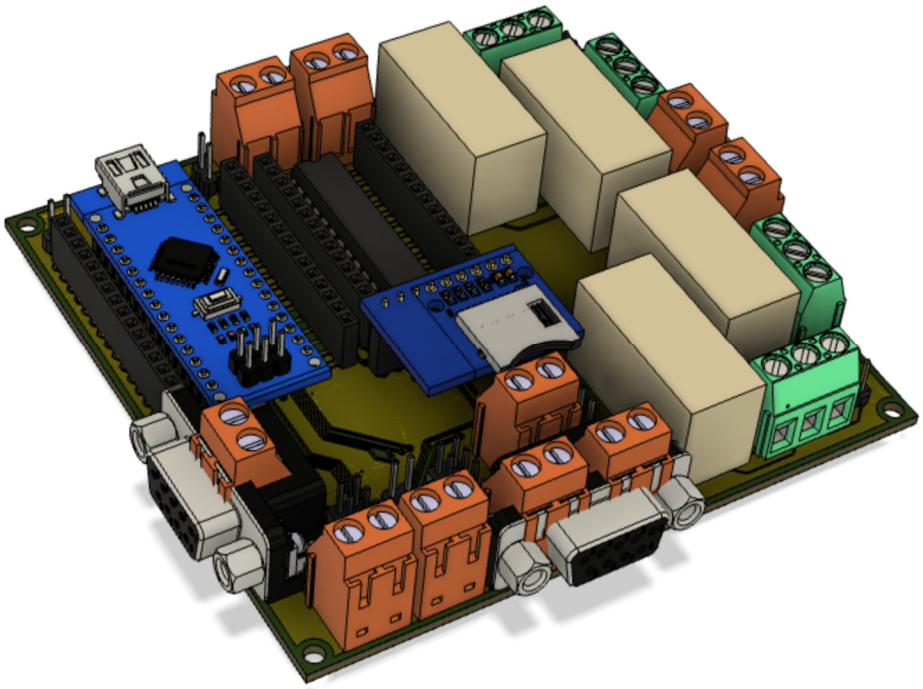


Figure 4-6: *Front 3D view of Ideal Gateway v2*

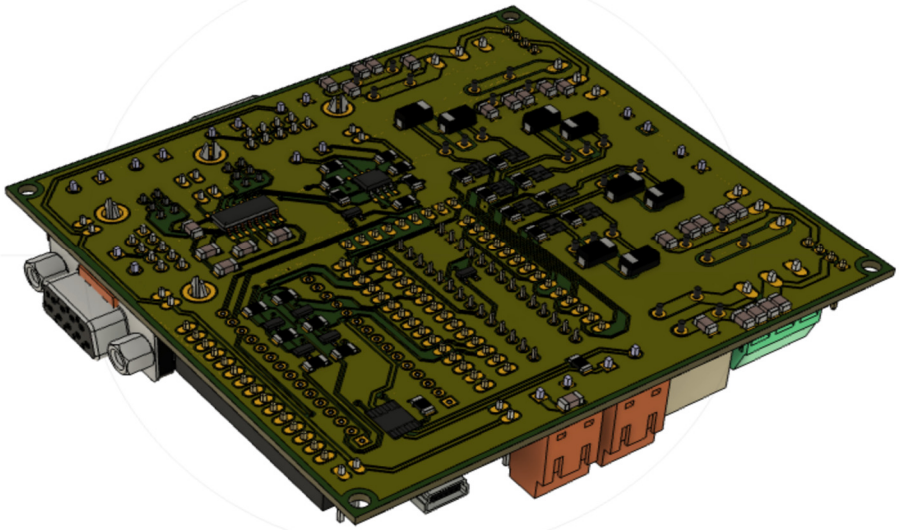
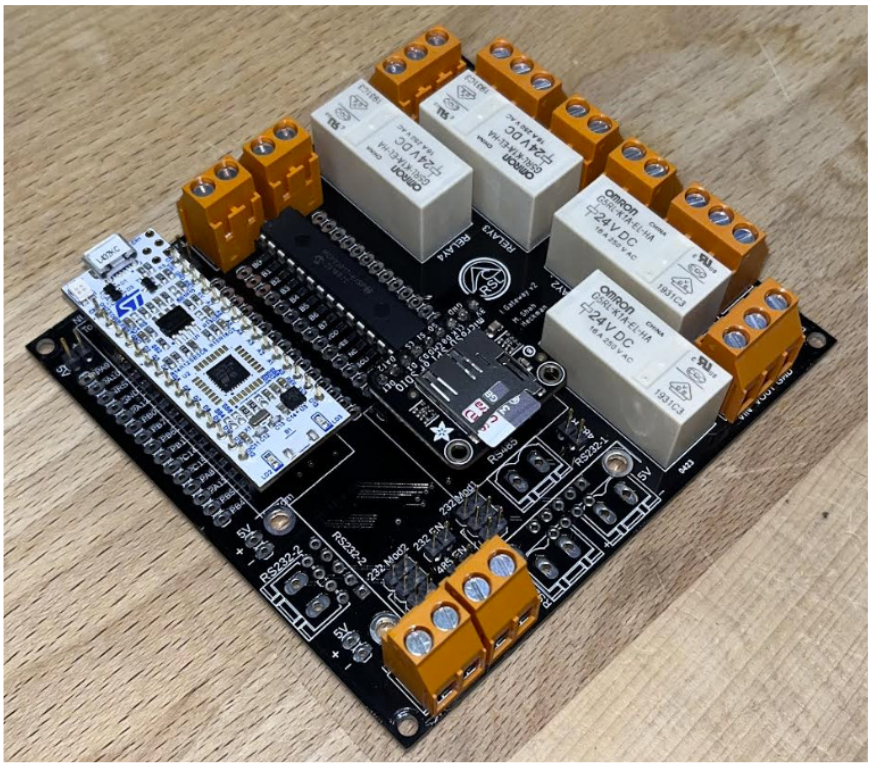
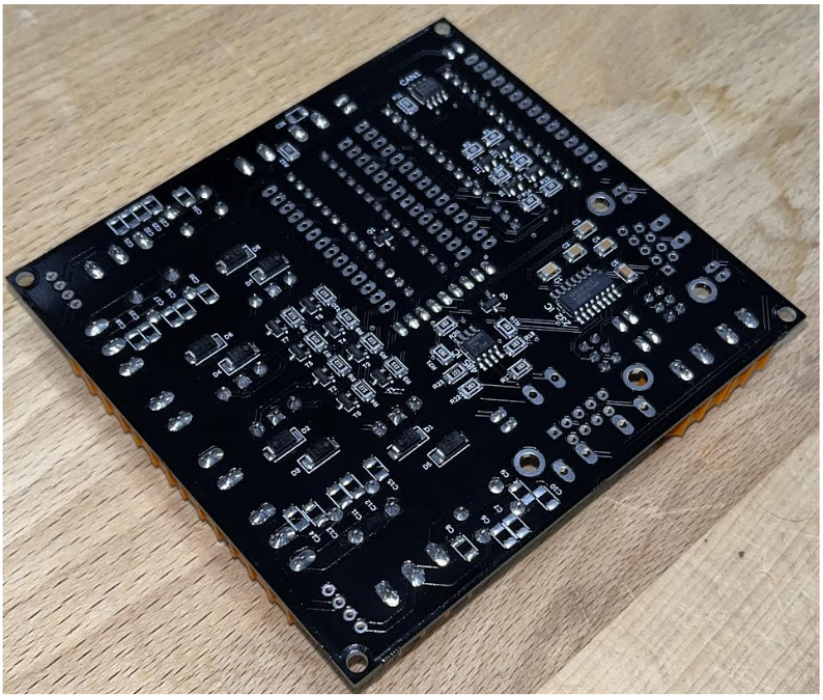


Figure 4-7: *Rear 3D view of Ideal Gateway v2*

We can integrate the mechanical development of the UAV and electrical development of the distributed node on the same software as it supports both CAD and CAE. This solution provides a plethora of testing and simulation options within the software.



a) Top View of Board



b) Bottom View of Board

Figure 4-8: *Final Implementation of the Designed/Fabricated/Stuffed Gateway Board v2*

5.0 Software Architecture

A significant amount of software development was conducted as part of this project. Custom software is executed on all of the distributed computing nodes developed for on-board control as well as on an off-board operator workstation used for interaction with the vehicle when it is in the local vicinity. This section describes the nature of the software to include a description of the layered architecture developed for the custom gateways described in Section 4, the general state machine operation of these gateways, the state machine that coordinates vehicle-level functional tasks, and the off-board workstation software.

5.1 Development Objectives

The software requirements included the need to support low-power operation of the processors, high-resolution power control of support components/peripherals, operation and configuration of peripheral equipment, local sequencing and control of peripheral equipment, communication to other processors and storage devices, etc.

In enabling these functions, an abstracted and modular composition approach was adopted. This made the software easier to manage due to design choices such as providing generalized interfaces to hardware, simple functional definitions promoting routine and rigorous test and verification, allowing software development to be parallelized across several development sub-teams, etc.

5.2 Distributed Gateway Software Architecture

Figure 5-1 shows the layered software architecture that has been adopted for the STM-based gateways. The lowest level controls microprocessor hardware interaction directly with attached components. Each higher level provides more abstracted and task specific functionality. This section describes the functions performed at each layer of the architecture, beginning with the lowest level and working up in abstraction

Level 0: The HAL (Hardware Abstraction Layer) Drivers are the fundamental “bit level” software that controls interaction with the hardware components that are connected to the processor. This includes hardware such as the components that implement SPI, I2C, UART, CAN, timers and timer functions, GPIO, etc. There is an extensive set of parameters that are supported by the HAL drivers in order to provide desired functionality to higher level functions.

Level 1: This functional block handles the power switching for different devices connected to the gateway. Power switching on the ideal gateway are of two types:

- High Power Channel Switching: High power/current loads are provided through relays. To switch these relays on/off, software selects the appropriate channel through the GPIO expander, which triggers transistors that, in turn, activate the relays in order to provide power to the connected device.

- Low Power Channel Switching: For lower power/current components, power switching takes place directly via the GPIO Expander, which triggers a transistor in order to enable power to each device. These lower power devices include the UART bridges, SD card, and other externally connected devices using low power.

For both high and lower power switching, the GPIO expander is used as described above. The GPIO expander is directed by the microcontroller via the I2C interface. Although power switching generally supports higher level functions, it is so fundamental to all levels of the software (above 0) that the decision was made to implement it as a Level 1 function.

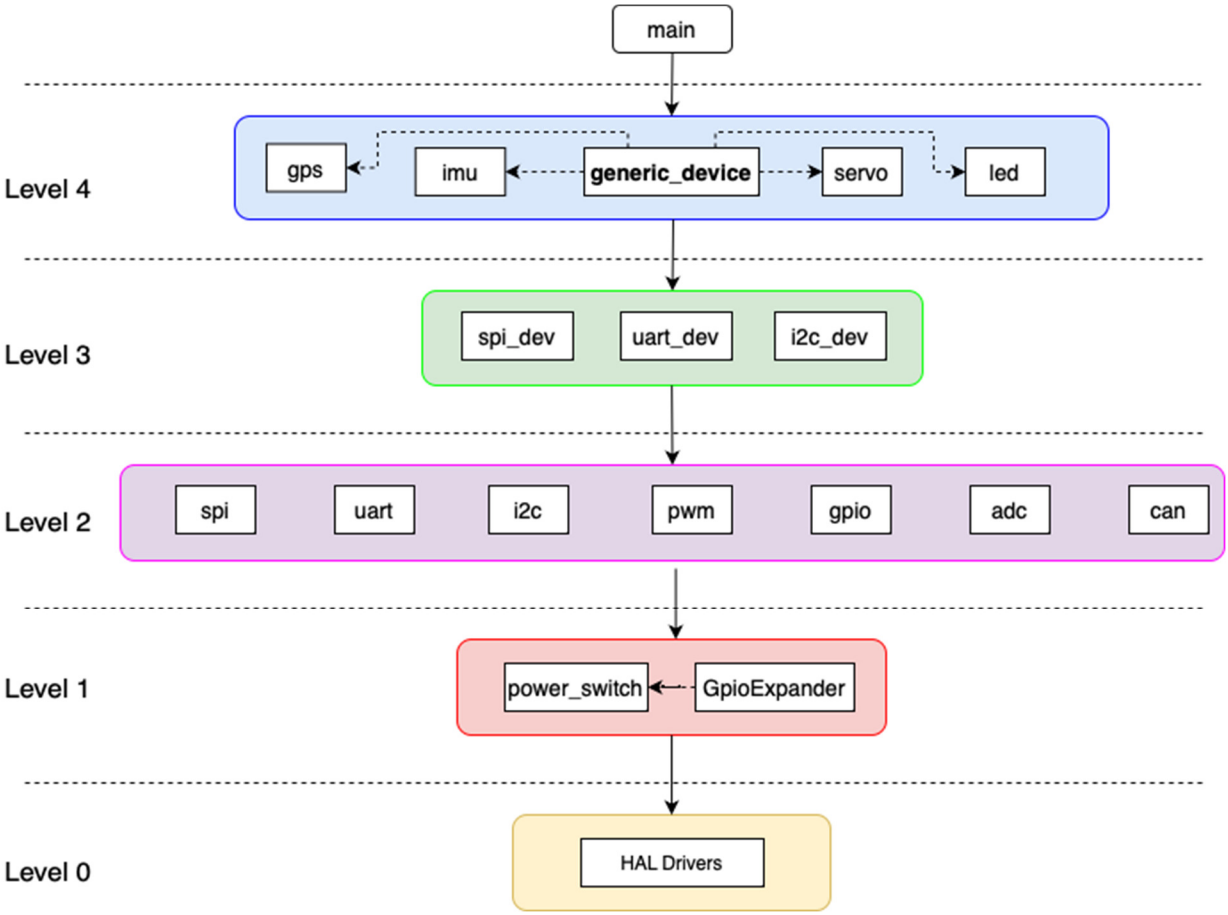


Figure 5-1: *Block diagram of the Layered Architecture*

Level 2: The level 2 functions implement various communication (I2C, CAN, SPI, UART) and hardware interface protocols (PWM, ADC, GPIO). All of these are implemented through manipulation of the processor's hardware pins, which are controlled by the Level 0 HAL driver software. However, these higher level functions are implemented given the need for abstraction. As such, the level 2 functions are specific to the functions we require, provide more user friendly naming conventions, and simplify the specification of HAL parameters given the

implementation of specific protocols. An example of the relative simplification that is achieved through the use of the level 2 functions is provided in Figures 5-2 and 5-3. Both functions implement the same functionality, with the raw HAL implementation shown in Figure 5-2 and the level 2 I2C function shown in Figure 5-3; clearly, the level 2 abstraction simplifies the programming task.

```
HAL_StatusTypeDef HAL_I2C_Master_Transmit(I2C_HandleTypeDef *hi2c, uint16_t DevAddress, uint8_t *pData, uint16_t Size,
                                          uint32_t Timeout)
{
  uint32_t tickstart;

  if (hi2c->State == HAL_I2C_STATE_READY)
  {
    /* Process Locked */
    __HAL_LOCK(hi2c);

    /* Init tickstart for timeout management*/
    tickstart = HAL_GetTick();

    if (I2C_WaitOnFlagUntilTimeout(hi2c, I2C_FLAG_BUSY, SET, I2C_TIMEOUT_BUSY, tickstart) != HAL_OK)
    {
      return HAL_ERROR;
    }

    hi2c->State     = HAL_I2C_STATE_BUSY_TX;
    hi2c->Mode      = HAL_I2C_MODE_MASTER;
    hi2c->ErrorCode = HAL_I2C_ERROR_NONE;

    /* Prepare transfer parameters */
    hi2c->pBuffPtr  = pData;
    hi2c->XferCount = Size;
    hi2c->XferISR   = NULL;
```

Figure 5-2: *HAL Function to send I2C data*

```
HAL_StatusTypeDef i2c_send(I2c *i2c_chn, uint8_t *data_tx) {
    HAL_StatusTypeDef ret;
    //Currently blocking code, try with interrupt later

    ret = HAL_I2C_Master_Transmit(i2c_chn->hi2c, i2c_chn->addr, data_tx, sizeof(data_tx), HAL_MAX_DELAY);
    return ret;
}
```

Figure 5-3: *Wrapper I2C function of level 2 to send data*

Level 3: This level defines the characteristics and particular parameters of the devices which use the interface protocols defined in the level below. Different devices use different pins of the microcontroller, might use different interface protocols, may need power switching capabilities from the gateway, etc. These specific details pertaining to the different devices connected to the gateway are defined in the Level 3 structures. The level 3 functions configure the new device connected to the gateway by calling the level 2 functions and creating separate structures for them.

Level 4: This layer is used by the user to program and configure the microcontroller with the specific components connected to it. For each component, it provides functions such as Initializing important parameters such as interface standard, giving specific power pins to that device, and creating operational functions on top of the device level functions made available in

level 3. Providing this level of software functionality increases the efficiency with which the user interacts with a gateway connected to a specific array of devices.

Main Program: Given the capabilities provided by the layered architecture, the MAIN program that runs on any gateway implements the overall gateway's application-specific functionality. It does this by implementing a state machine that dictates the subsystem's functional state, by interpreting commands that it receives via the CAN bus, and by using a system of processing flags to implement these commands. These functions are summarized here:

1. Gateway State Machine: The state machine manages the functional operations of the gateway. The ACTIVE machine state is the one in which standard gateway functions are implemented in terms of using subsystem equipment. Other states include SLEEP (low power state when not in use), INIT (startup state to configure the subsystem upon power up), DEBUG (for testing gateway functionality during debug/test/development activities), and EMERGENCY (for shutting down gateway operations during emergency conditions). The operation of the gateway state machine is described more fully in the next section.
2. CAN Message Processing: Gateways are networked via the CAN bus and must process arriving commands. Figure 5-4 depicts the general flow of CAN message interpretation, with this processing implemented in the MAIN program. When a CAN message is received, the microcontroller uses an interrupt and associated interrupt service routine to stop the program so that it can decode and execute the CAN message. Execution of the command is performed primarily through the manipulation of software flags that are used in commands that exist at a lower level in the gateway's software architecture and which explicitly manipulate the configuration and operation of subsystem components.

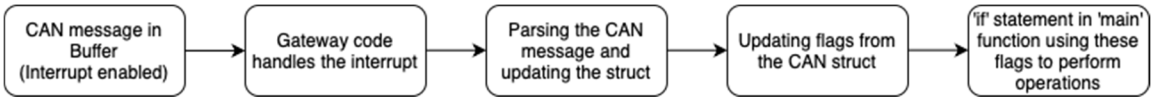


Figure 5-4: *CAN message translation in gateway*

3. Functional Flags/Variable Processing: The MAIN software is written using a set of flags to operate the devices connected to the gateway. For example, Figures 5-5 through 5-7 show excerpts of lines of code from the main block of actuator gateway. In this example, the flags are defined in a two-fold method. The flags in Figure 5-5 are the "parent" flags, which generally specify the high level functional processing for the gateway (e.g., "perform heading control"); in Figure 5-5, these flags are being initialized. Setting these high level functional flags leads to the manipulation of lower level "child" flags, which generally specify functionality for individual components (e.g., for a heading control parent flag, child flags turn on the required components within the subsystem to implement heading control). An example of this can be seen in Figure 5-6 and Figure 5-7; in this case, child flags are manipulated and are ultimately used to power and communicate with devices (e.g., the rudder servo and magnetometer, both used in heading control) and to configure and execute functional code within MAIN (e.g., turn on the heading control PID loop, etc.).

```
/* GATEWAY CONTROL VARIABLES - CHANGE AS NEEDED */
//All variables get set depending CAN messages. The
//actuator variables
bool heading_control_on_received = false;
bool heading_control_off_received = false;
bool thruster_control_received = false;
bool heading_control = false;
bool rudder_set = false;
bool thruster_control = false;
```

Figure 5-5: *Code block showing a few parent flags defined in the actuator gateway.*

```
case ACTIVE:

    if (heading_control_on_received == true){
        servo_power_on = true;
        magnetometer_power_on = true;
        heading_control = true;
        heading_control_on_received = false;
    }
```

Figure 5-6: *Code block showing how different flags are changed due to change in a master flag.*

```
if (magnetometer_power_on == true) {
    can_send(can_obj1, can_test_msg2, id_raspi);
    device__on(dev_magnetometer, false);
    magnetometer_power_on = false;
}
```

Figure 5-7: *Code block showing the process of switching on of the magnetometer device.*

5.3 State Machine for an Ideal Gateway

Given the software architecture described in the previous section, this section continues the discussion of the STM-based gateway state machine, describing it in greater detail.. From the point of view of application functionality, the gateway hosts a state machine that supports a number of different functional modes. The primary functional mode is the ACTIVE mode, in which application code is executed to operate the gateway's components in order to implement AUV-oriented functions, such as sensing, actuation, control, payload operation, etc. Additional functional modes are required for initialization of the gateway, power savings, debug operations, and handling emergency conditions.

Figure 5-8 shows the state flow for three specific functional modes. Upon power up, the gateway enters the INIT mode for initial configuration of the processor. It then enters ACTIVE

mode, in which it can perform gateway-specific applications. It can be switched between ACTIVE and SLEEP mode, with SLEEP being used when no applications are required. In SLEEP mode, the processor and its components are put into an ultra-low power mode state.

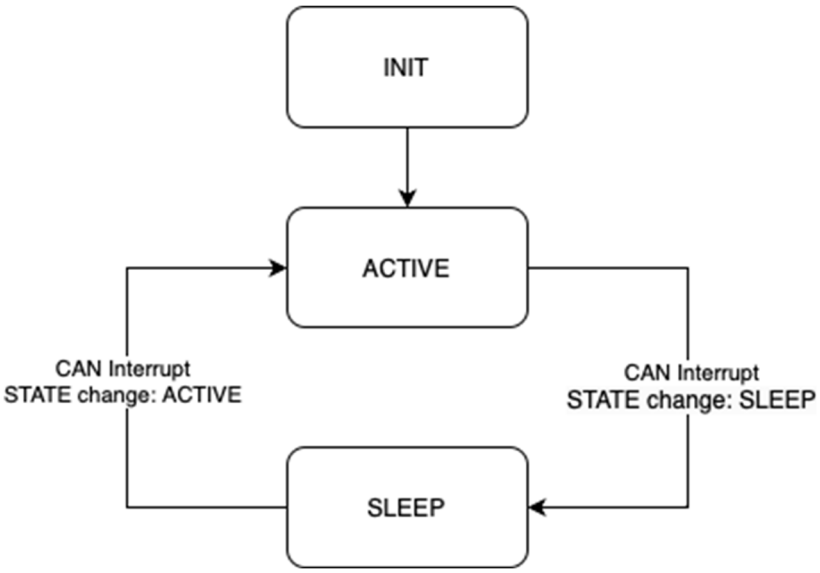


Figure 5-8: *State Machine diagram for Actuator Gateway*

INIT state: The INIT state software places the microcontroller into a known, benign state upon power-up. This includes initializing global variables, ensuring all power channels for external components are off, and possibly placing components in an initial condition (e.g., positioning of the rudder, etc.). The power off process is essential because, in the case when the gateway loses power abruptly, the relay state is maintained during a power cycle. This means that components that were on if power is lost will be powered back up upon processor power-up. This could create unwanted transients and an unknown power state. Therefore, all power channels are explicitly powered off as part of the INIT process.

ACTIVE state: The gateway is switched to ACTIVE in order to perform application-oriented tasks. This includes configuring/operating/controlling connected components to achieve tasks such as vehicle navigation, payload operation, etc. These high level application-specific tasks are programmed as part of the MAIN level of code, with functions exploiting lower levels of the hardware architecture to control power, communicate with components, etc. In the next section, an example of ACTIVE state functions is provided for a specific vehicle subsystem.

SLEEP state: When not in use, the gateway is put into the SLEEP mode via a command from the CAN bus. Before the microcontroller goes into sleep mode, the user can provide certain steps to perform before going into the low power mode. The steps may include turning some of the power-switching relays which may be powering devices that are not important in the sleep mode. This helps the AUV to conserve the battery for essential functions only.

In the low power M4 microcontroller, there are multiple sleep modes that it can be put into. For now, we are only using one specific microcontroller sleep mode which keeps some of the essential interface communications on such as the CAN interface. The gateway will receive the wake-up call through this CAN interface in order to switch into the ACTIVE state.

Two special states are also available:

- DEBUG state: This mode is used to test all the functionality of the gateway and its hardware. This state is not entered into as part of normal state machine processing. To invoke this state, the user is required to change the flag which governs the state machine, to exclusively proceed through the DEBUG state, in the main block.
- EMERGENCY state: This mode is activated when the gateway finds itself in a compromising state or it receives an external message from the CAN interface to switch to this state. In both cases, the gateway has to go into complete lockdown by power-switching everything off and waiting for an external, manual reset to start the operation again.

5.4 Actuator Gateway Example

Given the general description of the gateway state machine process, this section provides a specific example of how it is implemented. In this example, we consider what we currently call the Actuator Subsystem. This subsystem is responsible for AUV mobility within the horizontal plane, which is dictated by any of a number of mission-level navigation modes.

The Actuator Subsystem consists of a gateway computing node which is connected to the CAN bus from where it receives commands. The Actuator Subsystem gateway is connected to several actuator-specific components, to include the thruster for vehicle forward thrust, a rudder servo that is connected via a transmission to the rudder in order to influence vehicle yaw/heading, an Inertial Measurement Unit for sensing the vehicle's heading/direction, and an SD Card for data logging capabilities.

Operationally, the Actuator Subsystem can perform a number of functions as per commands via the CAN bus. These can be categorized into low-level component-specific functions and higher-level task-level functions:

- Low Level functions: There are component-specific functions like turning on/off components (the magnetometer, thruster and actuator), reading the heading from IMU, setting thruster speed, turning the rudder servo to a specific angle, etc.
- High Level functions: These are more complex functions which often blend the operation of multiple components, implement low level command sequencing, enable control loops, and so on. For example, one high level function is to implement heading control. To implement this command for a prescribed heading setpoint, a realtime control loop is established in which the IMU heading is read, a PID control compensation command is computed, and a rudder angle is commanded.

When powered on, the actuator gateway enters the INIT state. In this state, the gateway power switches off all the external devices, potentially centers the rudder servo, and configures the CAN bus for communication with external processors. The actuator gateway then enters the ACTIVE state automatically.

Once in the ACTIVE state, the actuator gateway accepts CAN messages with either low or high level function commands. Based on these, the actuator gateway decodes the CAN message, updates the functional flags and performs the functions based on the command's functional flags. If the command is to go to SLEEP, the gateway initiates the SLEEP state.

In a mission, an example of actuator gateway activity could be the following:

1. Power up and perform the INIT state functions to initialize the gateway, sending a confirmation message to the 'main' gateway indicating that it is ready to take commands, and entering the ACTIVE state;
2. Monitor the CAN bus for messages addressed from the 'main' gateway to the actuator gateway, and decoding those messages.
3. For messages with functional comments, the message flags are processed in order to perform functions such as powering components on/off, setting thruster level and rudder angle, reading the IMU heading, executing a heading control loop, etc.
4. When in the ACTIVE mode, if the message includes a command to enter the SLEEP mode, the gateway does so;
5. When in the SLEEP mode, if the message includes a command to enter the ACTIVE mode, the gateway does so.

5.5 On-Board Communications Gateway

Work to this point has focused on implementing the AUV's distributed computing system using only the custom-designed STM-based gateways, with one notable exception. Before detailing this exception, it is noted that one of the advantages of a distributed, networked computing system is that different computing nodes can easily be deployed throughout the system in order to meet the particular requirements of any particular subsystem. The different computing nodes can be implemented with different processors, be programmed in different languages, etc. It is the distributed computing interface standard that one adopts that allows this; in the case of the Huracan program, the interface presumes the use of a CAN bus for inter-node communications.

Given this, we elected to demonstrate this ability to mix and match processor nodes by incorporating a Raspberry Pi into the system. The Pi is able to interface with the CAN bus (as well as the power bus, the high data rate bus, etc.), thereby making it compatible with the rest of the distributed computing system. The Pi was chosen to serve as an external communication gateway with an off-board operator workstation. The Pi is well-suited to this task given the communication requirements that the team adopted:

1. First, we wished to support communication for some functions via a web-based interface; the Pi is able to host a web server and execute supporting scripts for interfacing server communications with the CAN bus.
2. Second, we wished to support streaming communications for joystick-based piloting; the Pi is able to host the streaming ROS (Robot Operating System) environment and to interface ROS communication node functions with the CAN bus.
3. Third, the Pi easily supports WiFi-based wireless communications, making it easy to wirelessly connect to a laptop-based workstation.
4. Fourth, unlike the STM-based gateways, the Pi had the computational power to support all of these services simultaneously, and to run the Linux operating system for enabling their execution.

Accordingly, the PI software consisted of a variety of off-the-shelf packages, to include the Linux OS, a ROS installation, and a web server. Elements of this software were customized appropriately, and custom scripts were written to integrate this software with the CAN bus, etc.

5.6 Operator Workstation and Interaction with the Vehicle

An operator workstation, which is physically distinct from the AUV system, is used by the operations team for a variety of functions: to configure an automated AUV mission, to drive the vehicle via joystick when it is local, to perform diagnostic and test activities, etc. A ruggedized laptop computer running Linux is used for the workstation, and it can communicate wirelessly via WiFi with the on-board Pi communication gateway.

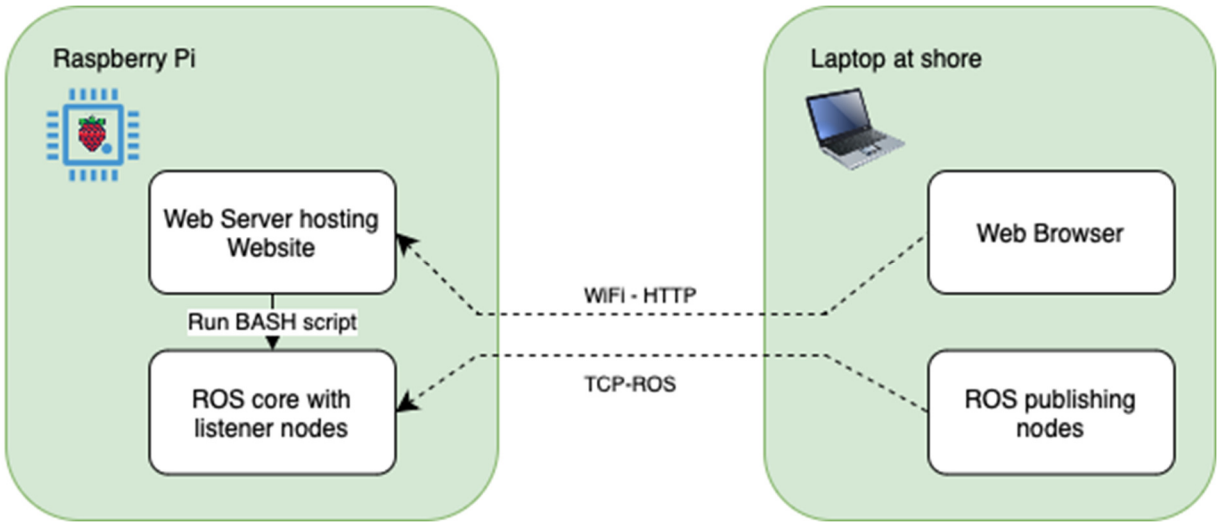


Figure 5-9: *Mission Sequencing block diagram.*

This section describes the software that implements interactions between the workstation and the on-board Pi communication gateway; the software implementation of this is represented in Figure 5-9. These interactions are implemented in three distinct Interaction Modes:

- The first is a “diagnostic mode” in which commands can be sent to the vehicle, with any responses displayed in response. This can be used to set parameters that configure an automated mission, check the operation and status of vehicle components, etc. For example, a command can be sent to read the heading from the IMU in the Actuator Subsystem. Commands are relayed to the appropriate vehicle subsystem with any resulting values returned to the workstation for display.
- The second is a “mission mode” in which a pre-stored mission is selected and (typically) autonomous vehicle operations commence as per the needs of the mission,
- The third is “piloting mode” in which realtime joystick commands are relayed to the vehicle in order to directly control the thruster, rudder, etc. This mode includes a mechanism to stop the vehicle (e.g., power off the thruster) in case real time communications are interrupted.

The software implementation and communication flow for these three Interaction Modes is different due to the nature and requirements of the types of communication executed in each of the modes. In Diagnostic mode, a client-server web-based architecture is used given its simplicity and the non-realtime nature of the interaction. In Mission mode, the vehicle is put into an operational mode in which the WiFi connection will not be required (for normal operations). In Piloting mode, realtime response is desired and so a streaming architecture using ROS has been implemented.

Only one Interaction Mode is active at any one time and is selectable from the operator client interface, assuming the vehicle is in WiFi range of the operator workstation. If it is out of range, the vehicle is either mobile due to being in an autonomous Mission mode, or it is in Diagnostic mode awaiting a WiFi connection. The individual Interaction Modes are further described in the following subsections.

Diagnostic Interaction Mode: This mode is used to send non-piloting commands to the vehicle and to display any resulting responses. It is used to configure operational parameters, test/verify/demonstrate equipment functionality, etc.

This mode is implemented with a web-based client-server architecture. A web browser on the workstation interacts with a web server that executes on the on-board Pi via HTTP/HTTPS protocols to send/receive communications. As commands (such as “set the thruster level to 50% speed”) are received by the server, a Python script composes the appropriate CAN command and transmits it on the CAN bus that is connected to the Pi. If any responses are generated, are received on the CAN bus, processed by a Python script, and communicated from the Pi’s web server back to the workstation’s client interface for display.

Snapshots of the web client interface are shown in Figure 5-10 and 5-11.

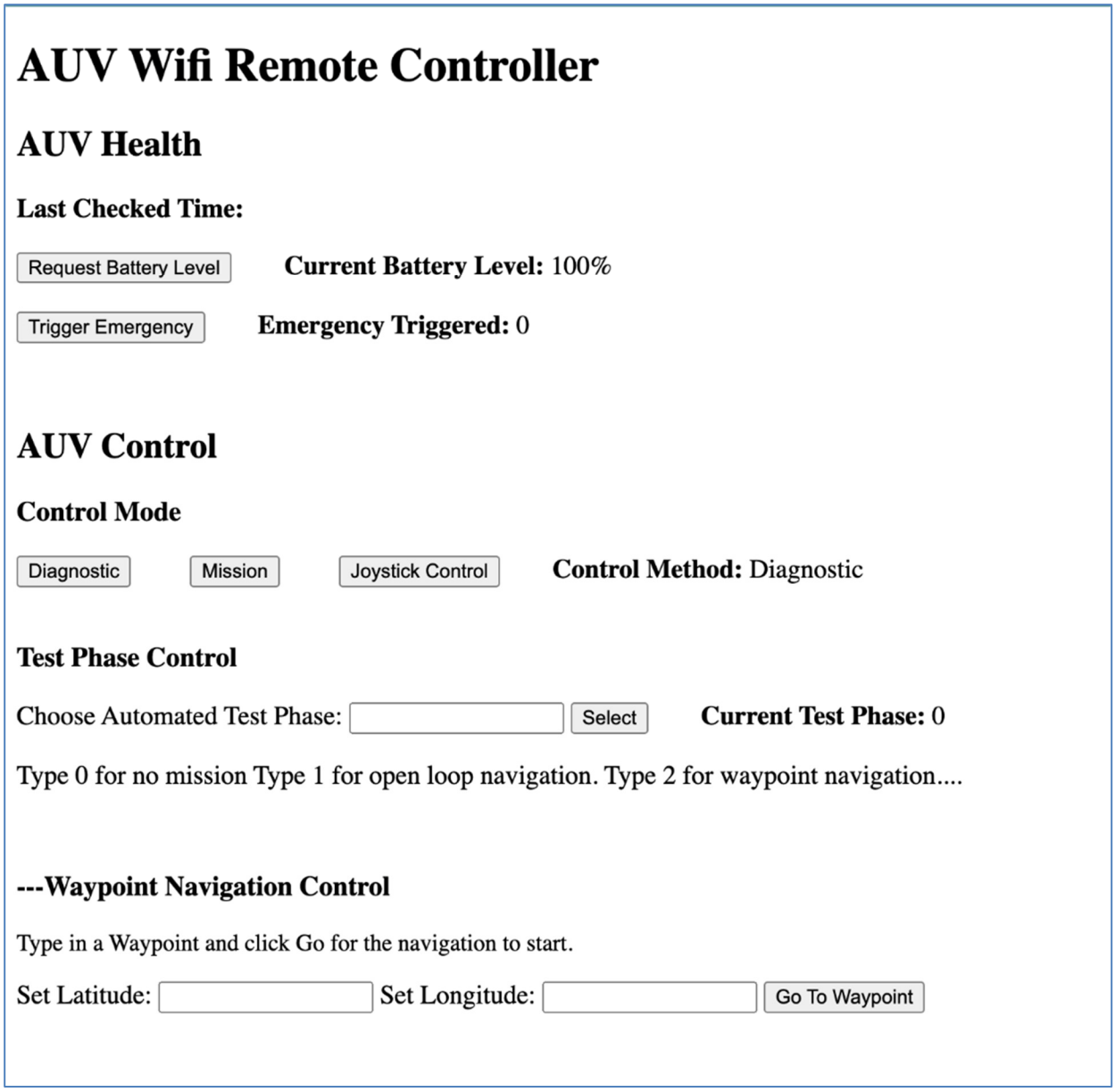


Figure 5-10: *Web interface showing controls to change to different modes*

Mission Interaction mode: This mode is used to enable the vehicle to conduct its own autonomous missions. These missions are pre-configured via the software residing on the mission management gateway. In general, any mission is implemented by running a state-machine that sequences and switches between mission modes as per the needs of the mission (as described in the Mission Level Operations section). As a result, composition of a mission requires the definition (programming) of the mission modes as well as a set of state transition criteria. Multiple mission profiles may be stored in the mission management gateway.

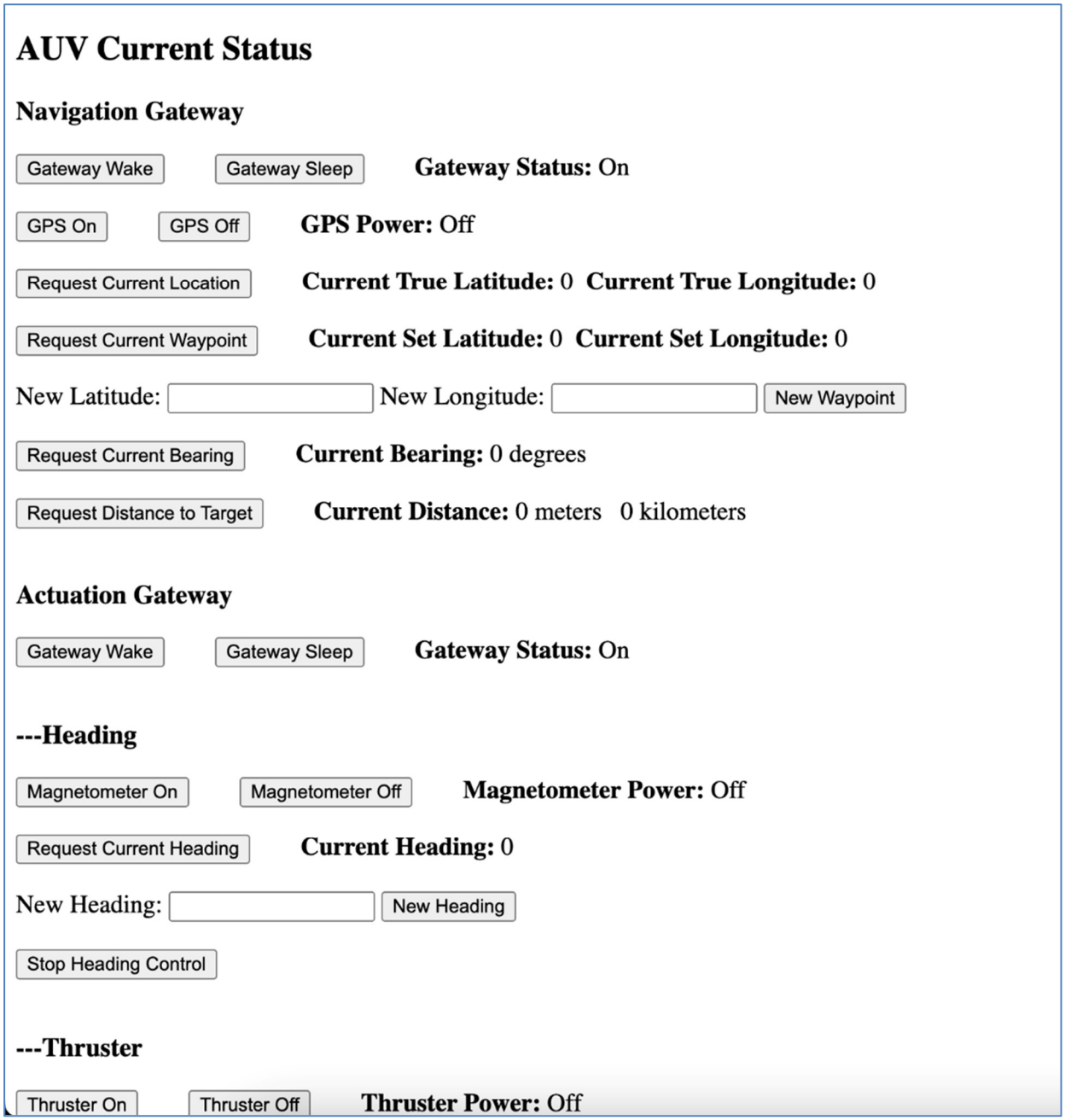


Figure 5-11: *Web interface showing controls for the Diagnostic Mode*

Given this, Mission mode allows an operator to specify and initiate a specific mission profile. It is initiated from Diagnostic mode by using the web interface to select a specific mission number for autonomous execution. Upon entering the mission number, on-board web server scripts communicate the mission number via the CAN bus to the mission management gateway, which then begins execution of the mission. At this point, the vehicle no longer requires the WiFi communication link (although it could presumably be used for things like status updates while the vehicle continues to be in range of the workstation). While the WiFi link, the operator may use the web client to terminate Mission mode and revert to Diagnostic mode.

Piloting Interaction Mode: Piloting mode is used to manually drive the AUV via a joystick connected to the operator workstation. In this mode, two streaming channels are established in order to send joystick commands and a heartbeat signal from the workstation to the on-board Pi. The streaming channels are configured using ROS (Robot Operating System) given its capability to implement distributed communications between processing nodes via TCP/IP, its ability to run on both the Pi and the workstation, its prevalence in the robotics community for robotic system prototyping, and its ease of use.

When in Diagnostic mode, piloting mode is entered by sending a command via the Diagnostic mode's web interface. Upon receipt, the Pi 's server uses a shell script to create two ROS *listener nodes*. Procedurally, an operator also manually starts a shell script on the Workstation in order to create two ROS *publisher nodes.* Once initiated, the pair of ROS *publisher-listener* nodes synchronize and are then able to communicate.

One *publisher-listener* ROS channel is used for joystick communications and relates joystick state data from the workstation to the Pi. This information is then converted to CAN messages and published on the CAN bus in order to operate the vehicle's thruster and rudder. This allows an operator to drive the vehicle on the surface when it is in close proximity to the workstation. This is used to maneuver the vehicle when local to the deployment vessel / dock / shore as well as when performing test operations. When in Piloting mode, Diagnostic mode can be re-established by sending a signal that terminates the Pi's ROS listener nodes.

The second *publisher-listener* ROS channel is used as a safety mechanism when in piloting mode. The risk is in losing communication connectivity while controlling the vehicle via the joystick, with the potential for the thruster to remain on without the ability to remotely control the vehicle. To address this, the second ROS channel sends a periodic 'heartbeat' signal from the workstation to the Pi. If the Pi's listener node stops receiving the heartbeat signal, it will issue a CAN message to disable power to the thruster (and potentially any other actuators) in order to terminate motion. In addition, Piloting mode is terminated, Diagnostic mode is re-established, and the ROS listener nodes on the Pi are disabled. When the WiFi link is reestablished, the Pi will be in Diagnostic mode, ready to receive commands from the workstation's browser. Joystick mode can be re-established at this point by initiating it via the standard approach.

5.7 Mission Level Operations

Although explicitly exploring mission operations was beyond the scope of this initial grant, the distributed computing system's hardware and software are ultimately needed to support such operations. For that reason, we have developed a framework for mission-level functions in terms of their software execution.

In previous sections, the STM-based gateway software architecture and state machine processing was described, the functionality and software services of the Pi communication

gateway was described, and the operator workstation - vehicle Interaction modes were described.

In this section, we describe the overall approach to conducting an operational mission with the AUV. These typically begin (and end) with operator workstation - vehicle interaction and then transition to autonomous vehicle operation. During autonomous vehicle operation, the vehicle performs a set of 'mission modes' that, together, implement the mission and which integrate the operation of multiple on-board gateways/subsystems. For example, a very simple candidate mission might include the following scenario:

1. Pre-deployment activities: The vehicle is on and being configured and checked out on deck or in the water at the deployment site, prior to embarking on the mission; in this mode, the mission sequence is uploaded, status checks are made, etc.
2. Joystick Piloting activity: The vehicle uses this mode such that the operations team can safely maneuver the vehicle away from the shore / dock / support vessel.
3. Surface Navigation activity: the vehicle navigates on the surface, using GPS and IMU/heading data as well as thruster and rudder operation; it uses this mode to navigate to a local point appropriate for diving.
4. Underwater Navigation activity: the vehicle navigates underwater, using depth and IMU/heading data (and possibly an altimeter) along with position estimators in order to move through the use of the thruster, rudder, and mass-slider system for pitch control; it uses this mode to navigate to an underwater location of operation.
5. Underwater Mow-The-Lawn activity: the vehicle executes mow-the-lawn maneuvers at several depths, taking data with its payloads in order to create an volumetric estimate of the environment;
6. Underwater Navigation activity: The vehicle uses this mode to return to the recovery area, surfacing at a safe standoff point.
7. Surface Navigation activity: The vehicle uses this mode to move into position ready for retrieval.
8. Joystick Piloting activity: The vehicle uses this mode such that the operations team can interactively maneuver the vehicle for retrieval.

In conducting these activities, the Diagnostic Interaction mode is used for the Step 1 pre-deployment activity in order to support vehicle checkout, testing, etc. The Piloting Interaction is used for Step 2 to move the vehicle into position on the surface from which autonomous operations are to be initiated. Steps 3-7 are conducted by entering Mission Interaction mode such that vehicle operations are managed by an on board Mission Management Gateway. Finally, in Step 8, with the vehicle back in WiFi range of the workstation, the Piloting Interaction mode would be established.

During autonomous vehicle operations, the Mission Management Gateway (an STM-based board) executes the specified mission, which consists of sequencing and managing a suite of autonomous mission states that define the mission. The mission management state machine is

defined by the set of possible autonomous mission states as well as the policy to transition between these states:

- Any particular autonomous mission state might consist of a navigation and/or payload objective. In the simple given scenario, activities 3-7 would be implemented as autonomous mission states. Of course, more complex mission states are possible. For example, the craft may sleep or drift for periods of time to save energy, then wake up to check conditions, and then based on those conditions either returning to a sleep/drift state or perhaps initiate a mission-oriented data collection state.
- Transitions between states can be established based on timers, sensed conditions, completion of specific tasks, etc.

For any specific autonomous mission states, multiple gateways work together to support the overall functionality. The Mission Management Gateway coordinates this interaction given the required functionality. For testing purposes, we have several of these to explore this approach, demonstrate application-oriented capability, refine the interaction among computational gateways, and so on. Reports on the successful demonstration of these modes are provided in the test section of this report. These test autonomous mission states include:

- Surface GPS Waypoint Navigation mode: In this mode, the vehicle performs (surface / horizontal plane) navigation in order to move to a designated location. To achieve this, the Mission Management Gateway reads position data from the GPS Gateway, computes waypoint-based rudder setpoints for a given thruster setting, and sends those rudder and thruster commands to the Actuator Gateway.
- Mow-The-Lawn Navigation mode: In this mode, the vehicle is directed to perform a mow-the-lawn maneuver by moving in designated direction for a specified period of time, then turning to move in the opposite direction with a lateral position offset. This process continues for a specific number of "laps." A simple version of this has been implemented given testing limitations. In this version, linear translation is conducted under heading control with the end of a transect designated by a timer. To implement this, the Mission Management Gateway commands the Actuator Gateway to translate with a specific thruster setting while controlling to a heading setpoint. Once a translation timer expires, the Mission Management Gateway implements a turn and then re-established heading control in the opposite direction, both via new commands to the Actuator Subsystem. A future version of this will implement path control using inertial estimates of lateral motion to account for current disturbances, etc. A more complex version will also enable depth control such that the maneuver is conducted at a specific depth; the current version does not take this into account since the scope of the project does not include field operations at depth.

6.0 Power Storage Design

Although development of a full power system will occur in a later Phase, for Phase 1 we explored the design of an appropriate power storage and management solution appropriate for the objectives of the Huracan program. The result is an intelligent battery system capable of monitoring and controlling power flow from an array of commercially available smart battery packs. In addition, this battery management system can communicate its status and faults with via the CAN bus, effectively acting as the computational front-end for the power storage system. Modularity and ease of maintenance were important features for this initial prototype and have been incorporated into its design.

6.1 Battery Selection

The Inspired Energy PH2059HD34 battery, shown in Figure 6-1, was selected for the Huracan power prototype for a number of reasons. This battery is a smart Li-Ion product that uses 8 high capacity cells in series to maximize its energy density and which specifically designed to be suitable for mobile vehicles. The battery nominally runs at 28.8V and 6 Amps with an energy capacity of 98 Wh. The battery includes its own over-charge, over-discharge and short circuit protection. For the future Huracan power storage system, a large number of these batteries will be packaged in parallel into a glass sphere, with several glass sphere units also wired in parallel. For Phase 1 prototyping, a small number (4, typically) of these batteries have been wired in parallel to explore wiring, control and packaging issues.

Figure 6-1: *Inspired Energy PH2059HD34 Battery*

6.2 Custom Battery Monitor/Control Gateway

The energy storage system, comprised of battery banks, provides power to all computational nodes in the distributed system via a standard power bus. Control of the energy storage system

is performed by one of the gateways that is connected to the others via the CAN bus. Because of the special requirements for power monitoring and control, however, this custom board is tailored to those tasks and is a separate design from the standard Ideal Gateway described in Section 4.

The power gateway uses the same low power ARM Cortex-M4 as the Ideal Gateway. It is responsible for communicating over SMBUS with each Inspired Energy smart battery pack, logging desired data in a microSD card, power switching the battery output to the power bus, and interacting with the craft's other systems over the CAN bus.

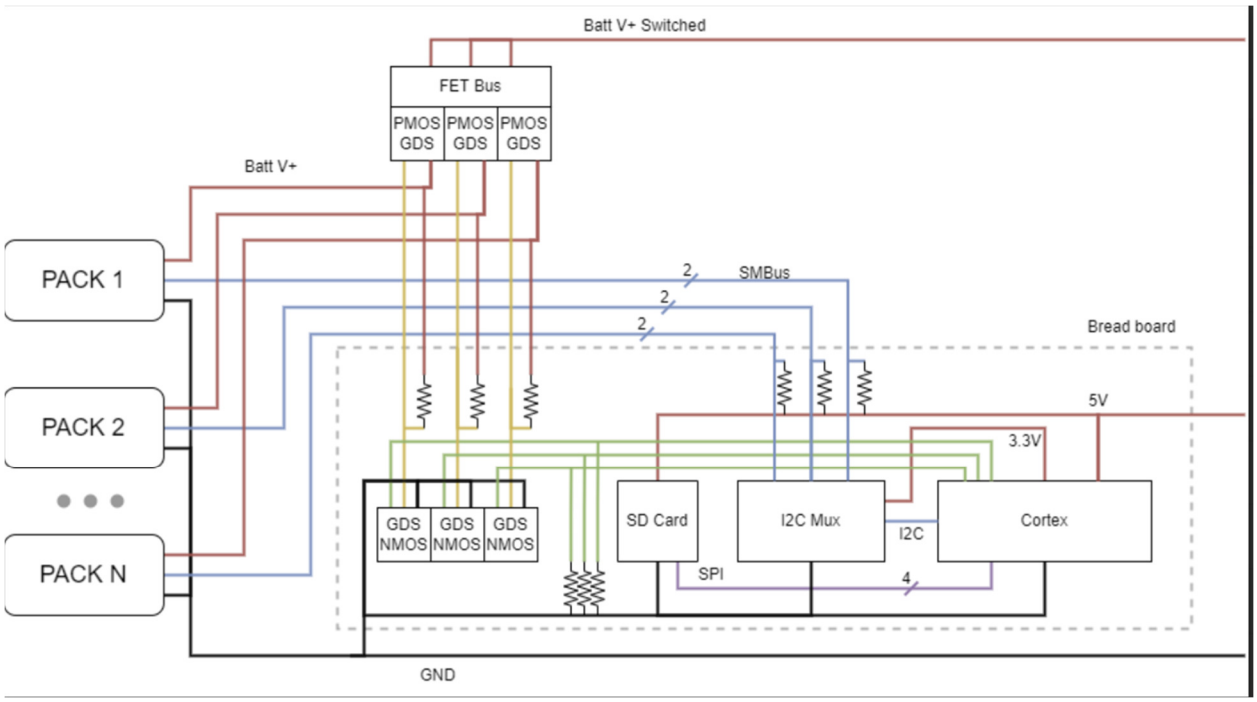


Figure 6-2: *Simplified diagram of power control node*

A notional block diagram of gateway components can be seen in Figure 6-2. Features of the design include the following:

- The Cortex processor provides system monitoring and control functions and is shown in the lower right hand portion of the Figure;
- Batteries are the "packs" shown on the left side of the diagram;
- The Cortex communicates with the batteries in order to access their status data via their SMBus, which is read by the Cortex over an I2C channel;
- The Cortex uses an I2C Mux in order to select which battery pack SMBus to monitor at any given time;

- Battery output voltage pins are connected to the Battery bus (at the top of the diagram) through the FET Bus, which is effectively a high power suite of switches; these FETs are activated by a lower poser bank of NMOS switches that are directly commanded by the Cortex in order to select which batteries should be providing power to the vehicle at any given time;
- The Cortex can log data to an SD Card via an SPI channel.

The SMBus/I2C integration implementation is worthy of note. Because the Inspired Energy battery packs use the SMBUS 1.1 revision there is no support for address resolution and a hardware solution was implemented in the form of an I2C mux. This allows a single two-wire interface on the M4 to communicate with up to 64 battery packs that share an identical, hardcoded address. In our initial prototype implementation, only two battery packs, out of a maximum of eight, are designated to each I2C mux.

The system designed is modular with the ability to daisy chain “daughter boards” for integrating additional batteries. A “mother” unit has all of the elements depicted in Figure 6-2. A “daughter” unit, however, connects to a mother via GPIO lines and an SMBus interface and only requires the I2C mux and switching hardware in order to add two more battery packs to the system at large.

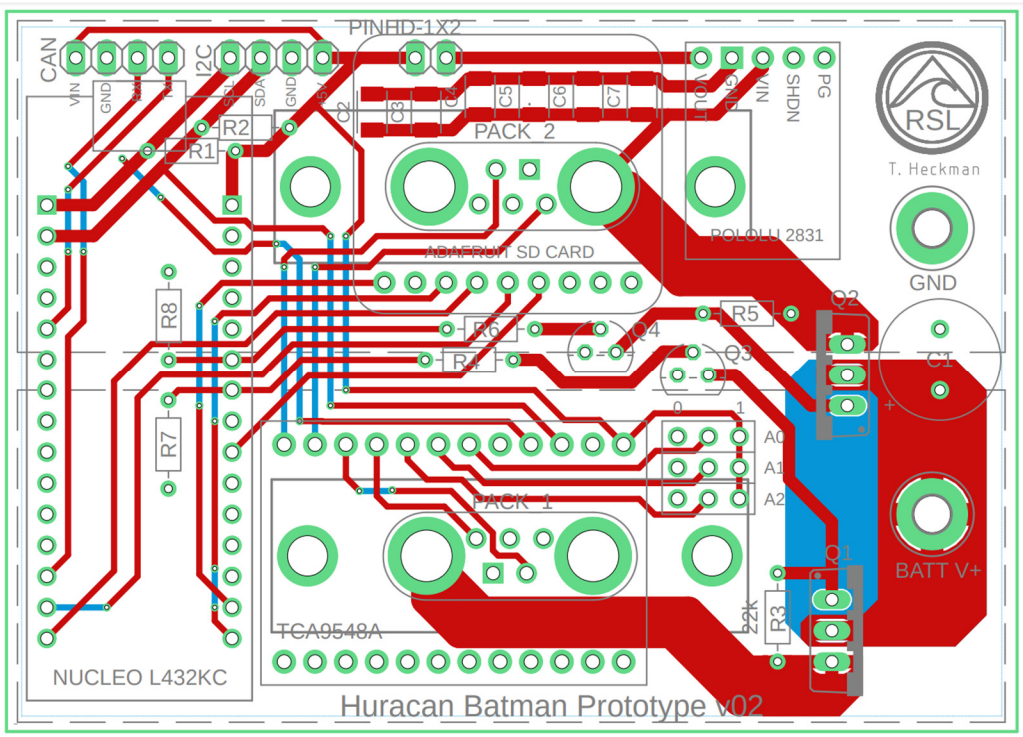
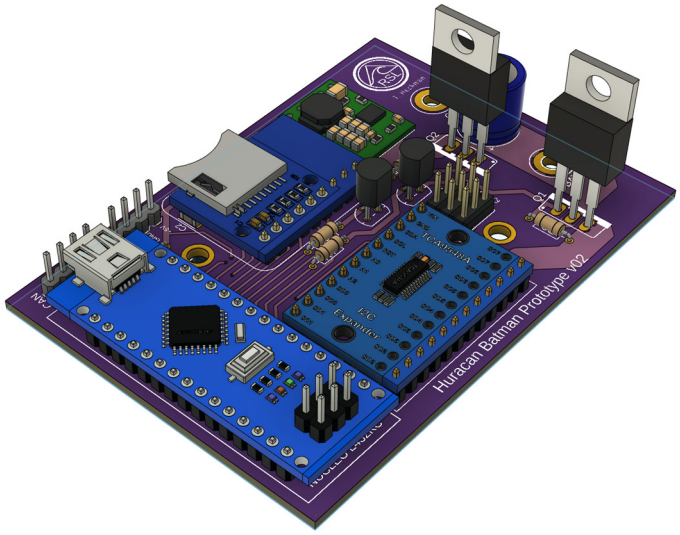
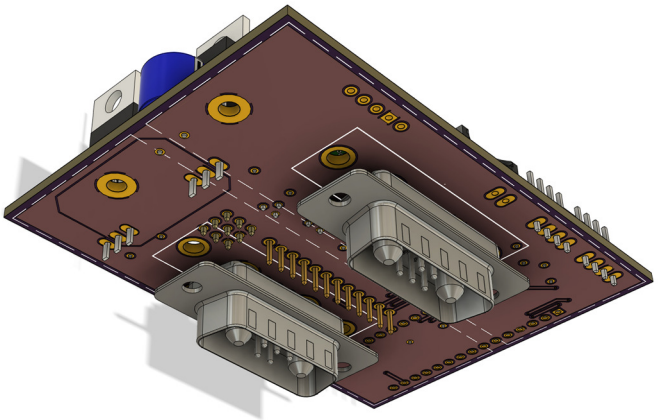


Figure 6-3: *Custom Battery Monitoring and Control Gateway PCB Layout*

Given the depicted block diagram, a PCB design of the battery management gateway was developed and has been used with great success during the test phase. Figure 6-3 shows the PCB board layout. The board was sized such that its two battery packs mount to the underside of the board with an eighth inch gap between them for a support structure. This makes the overall dimensions of the system similar to that of other lithium ion battery packs and allows for a range of packaging options when adding multiple daughter units. All components on the board (minus decoupling capacitors) are through-hole for ease of assembly. Figure 6-4 shows a 3D rendering of the PCB board. Figure 6-5 shows the fabricated and functional board, and Figure 6-6 shows the board packaged with its two batteries.



a) Top View of Board



b) Bottom View of Board

Figure 6-4: *3D Rendering of Battery Monitoring and Control Gateway Board*

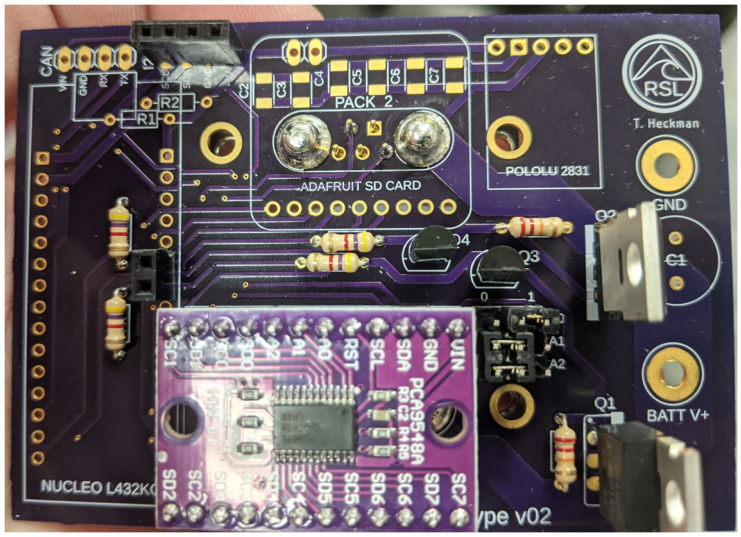


Figure 6-5: *Fabricated Battery Monitoring and Control PCB*

Figure 6-6: *Mother/Daughter Battery Monitoring and Control PCBs Each with their 2 Batteries*

7.0 Initial Analysis of Vehicle Dynamics and Control

Although development of a vehicle guidance and control system is planned for a future Phase, we developed a basic dynamic model of an AUV and designed simple controllers in order to inform requirements that might be appropriate for development of the distributed control system and the sensors/actuators that will be required for Huracan. Much of this work is documented in a separate project document; key elements of this work are summarized here.[9]

For this analysis, we started with simple but common assumptions such as the use of a linearized model, uncoupled pitch/depth and heading dynamics, etc. And since our initial physical tests are using our small test AUV, we are using physical parameters appropriate for the craft. In the future, these numbers can be easily updated for Huracan, and eventually a more complex nonlinear, coupled model can be adopted.

In a later phase, these models will be critical to conduct appropriate design trade-offs involving the mass slider (mass to be moved, range of motion, speed of response, etc.), the need for an elevator, the design parameters for the rudder (area, range of motion, etc.), sensor parameters (range, resolution, time response, etc.), power requirements for maneuvers, and more.

7.1 Pitch/Depth Control

The core element of the pitch/depth control system is the mass slider actuator, which moves an internal mass in order to change the longitudinal offset between the center of mass and center of buoyancy. This in turn creates a pitch torque that tilts the vehicle nose up/down, thereby allowing changes in depth. For the ultimate Huracan vehicle, the masses will be battery spheres with a motion control system operating in a wet environment; for initial prototyping, the mass is deadweight on an internal mechanism.

The transfer function for pitch angle as a function of mass-slider position is provided in Equation 7.1:

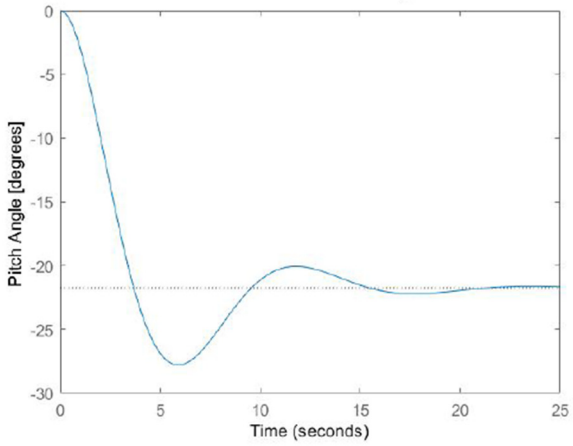
$$G_\theta(s) = \frac{\theta(s)}{x_m(s)} = \frac{\frac{M_m}{I_{yy}-M_{\dot{q}}}}{s^2 - \frac{M_q}{I_{yy}-M_{\dot{q}}}s - \frac{M_\theta}{I_{yy}-M_{\dot{q}}}} = -\frac{0.7866}{s^2+0.4343s+0.3316} \tag{7.1}$$

where theta is pitch angle, x_m is slider position, the I's are moments of inertia, M_theta is the pitch restoring torque (hydrostatic moment), M_q is the total damping for the vehicle (to

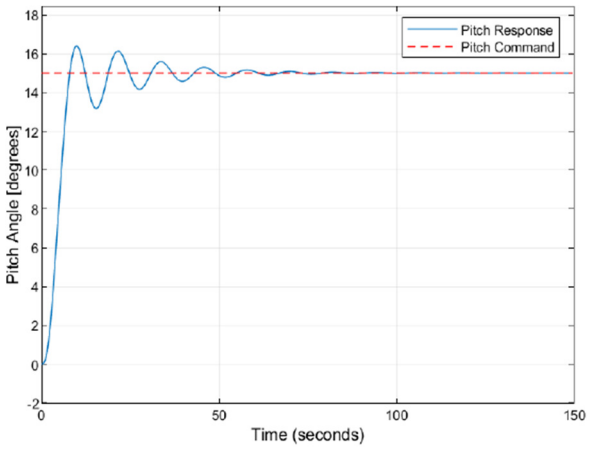
[9] R. Konrath, E. Kennar, J. Taylor, A. Knaus, S. Kidd Meyers, D. Guitierrez, "The Modular Oceanic Autonomous Underwater Vehicle for Novel Actuation (MOANA) 2.0," Senior Capstone Report, Adv. C. Kitts, June 2022.

include cross flow drag, added mass, and horizontal tail fin lift), M_m is the moving mass ratio with respect to the vehicle mass, and M_qdot is the inertial contribution of water that moves due to vehicle rotation.

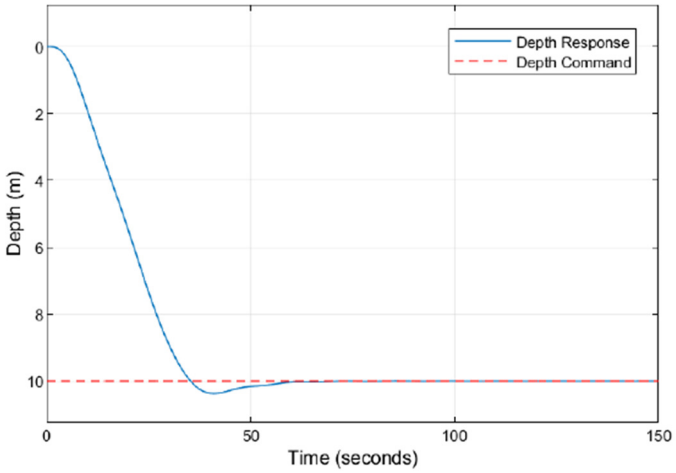
Figure 7-1 shows a step response of pitch for a given slider mass relocation; again, while numerical evaluation has been performed using parameters for our test AUV, the motion is faster but still characteristic of what we would expect for Huracan in terms of the nature of the response.



a) Pitch Step Response



b) Controlled Pitch



c) Controlled Depth

Figure 7-1: *Characteristic Pitch Dynamics and Control Using an Internal Mass Slide Actuation Strategy - Results are for a simple linearized decoupled analysis and using numerical parameters for the small-scale Huracan test AUV*

In terms of simple hardware prototyping, Figure 7-2 shows the small scale mass slider system for the small test vehicle. Using this and other prototypes, we have successfully shown mass position control as well as vehicle pitch control. Figure 7-3 shows how this unit was able to successfully control overall vehicle pitch for a 5 degree step command during a static test tank test.

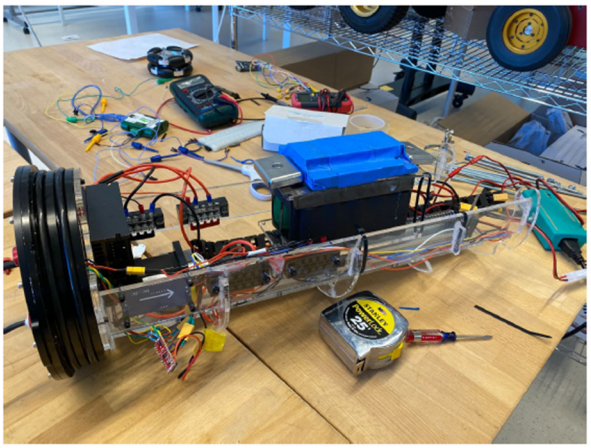


Figure 7-2: *Simple Mass Slider Actuator Prototype*

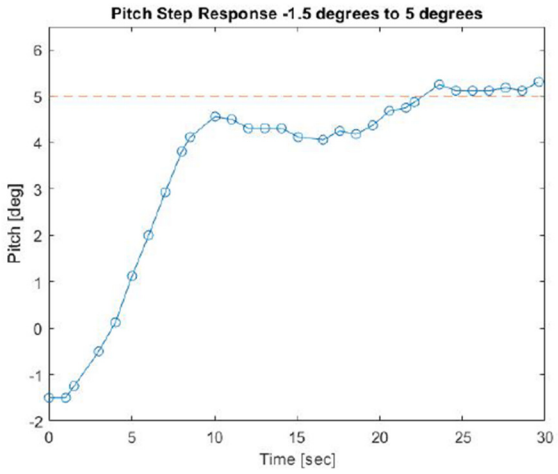


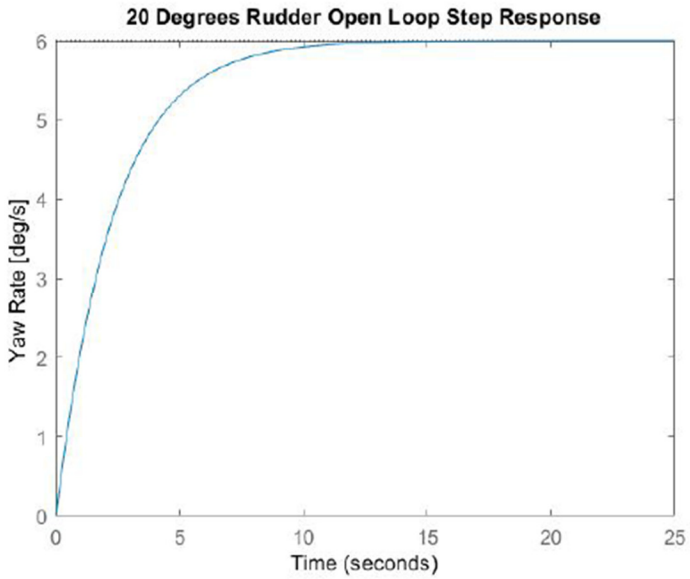
Figure 7-3: *Experimental Pitch Control Step Response*

7.2 Heading Control

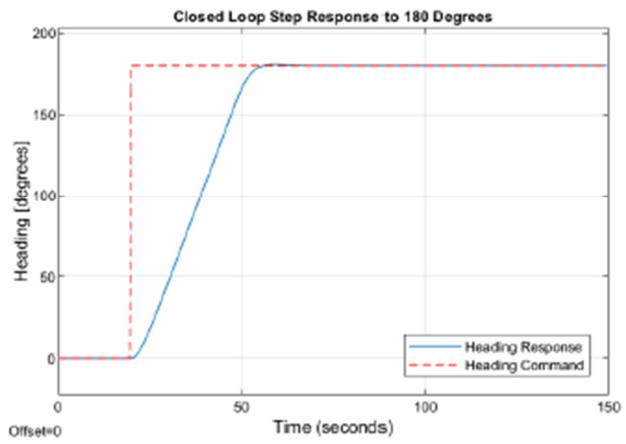
The core element of the heading control system is a conventional servo-driven rudder. Turning the rudder creates a yaw torque on the vehicle, which in turn turns the heading in the horizontal plane therefore altering the vehicle's heading. Equation 6.2 provides the transfer function for yaw rate given a rudder deflection,

$$G_{\dot{\phi}} = \frac{\dot{\phi}(s)}{\delta_r(s)} = \frac{\frac{1}{2}0.0433\rho v^2 S x_r}{\frac{I_{zz}-M_{\dot{q}}}{-M_q}s+1} = \frac{0.3}{2.316s+1} \tag{7.2}$$

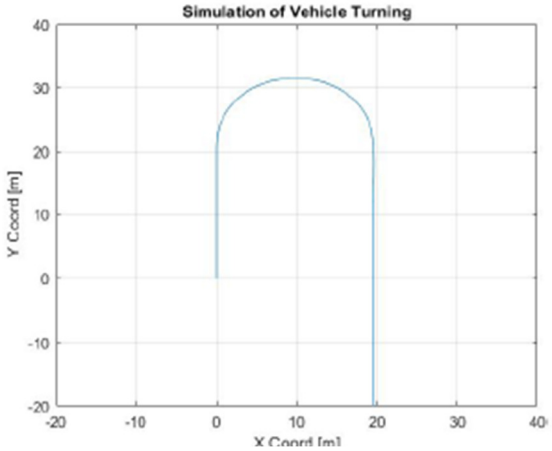
where phi is heading, delta_r is rudder angle, I_zz is the yaw moment of inertia, rho is water density, v is vehicle speed, x_r is rudder longitudinal station, and M_q and M_qdot are as previously defined. As with the pitch/depth analysis, numeric evaluation used parameters for the small test AUV but can be updated when necessary with parameters appropriate for Huracan once estimates for those values have been generated in future Phases of the program.



a) Yaw Step Response



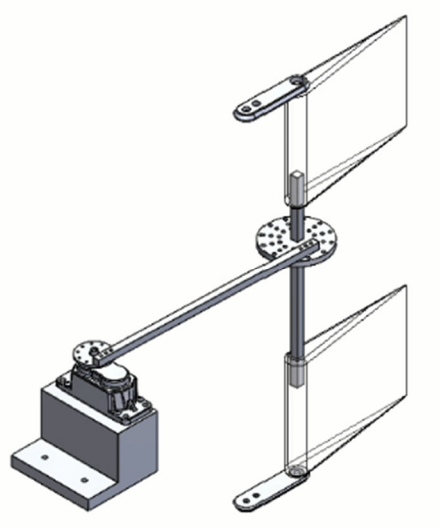
b) Controlled Yaw Step Response



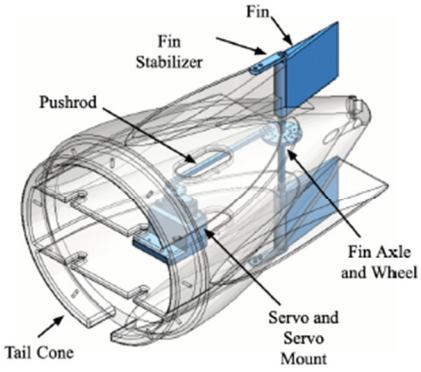
c) Controlled Turning

Figure 7-4: *Characteristic Yaw Control Using a Standard Rudder Actuation Strategy - Results are for a simple linearized decoupled analysis and using numerical parameters for the small-scale Huracan test AUV*

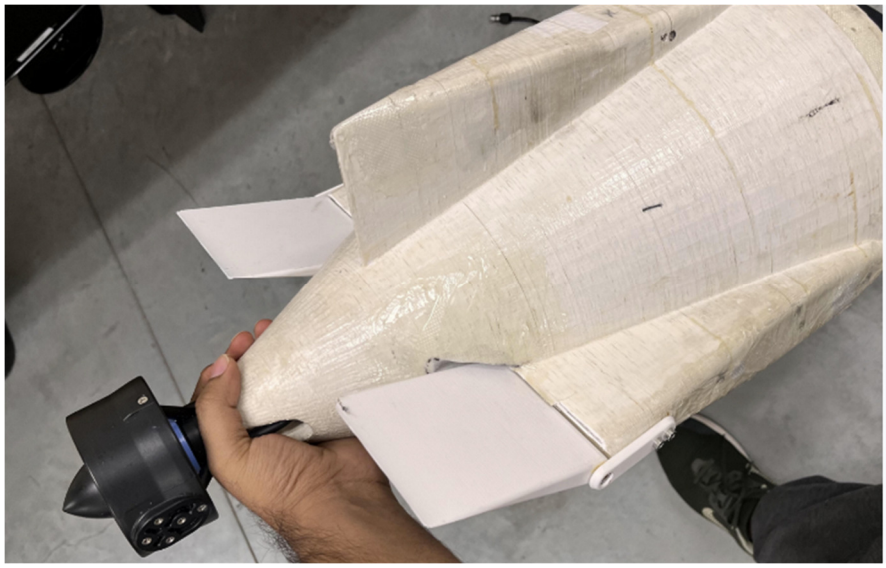
Using simple P control, Figure 7-4 shows how yaw control is achievable. Figure 7-5 shows design and prototyping of the rudder and tailcone assembly, used for yaw control of the small scale AUV prototype.



a) Rudder Transmission



b) Tailcone Design



c) Implemented Tailcone Assembly

Figure 7-5: *Rudder Actuation Implementation for the Small AUV Prototype*

7.3 Simple Vehicle Navigation

Given the controllers for pitch/depth and heading control described in the previous subsections, we explored simple state-based control of the vehicle for a candidate navigation maneuver. Specifically, we developed a tiered “mow-the-lawn” maneuver, which is certainly a mission-level navigation maneuver that will be used for Huracan deployments. As seen in Figure 7-6, a state machine sequences the set points for depth and heading control in order to dive to

an initial depth, mow the lawn within that plane while holding depth, and then repeating this process at other depth setpoints. We note that mow-the-lawn testing has also been demonstrated with the implemented hardware as described in the next section of this report.

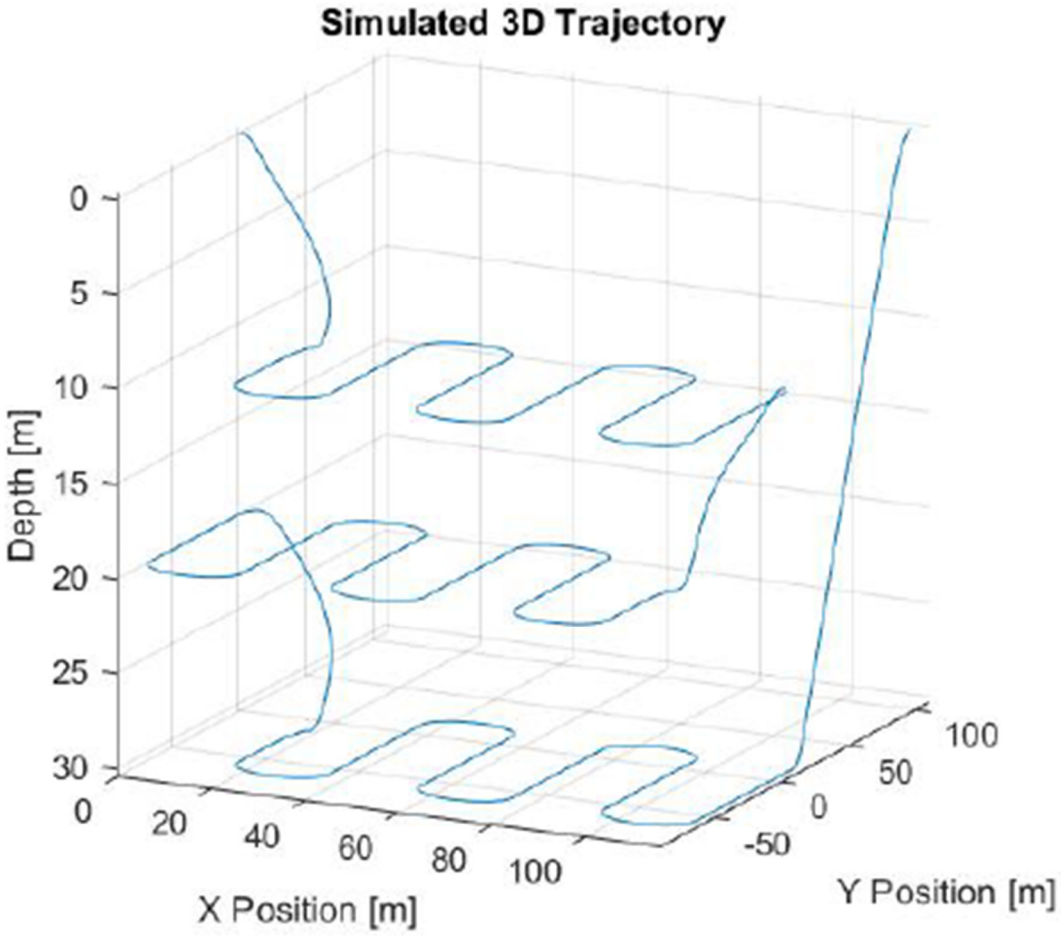


Figure 7-6: *Simulated Depth-Tiered Mow-The-Lawn Navigation*

8.0 Mission-Level Simulation/Test Activities

Although the team has not been given any specific mission objectives or requirements, we have been developing the system to provide a wide range of general services we believe will be necessary for the class of long-duration missions under consideration. From a mission perspective, we are assuming the need for a wide range of operational tasks to include functions such as the following:

- The ability to perform low level system checks and tests when physically connected to the system;
- The ability to perform system checks and higher level diagnostics when wirelessly connected to the system but still in proximity (e.g., with vehicle nearby in the water prior to initiating a mission);
- The ability to log data as appropriate;
- The ability to pilot the vehicle on the service via a local wireless piloting interface, such as a joystick controller;
- The ability to perform surface and subsurface waypoint navigation;
- The ability to perform surface and subsurface waypoint-defined path navigation;
- The ability to perform higher-level navigation maneuvers such as “mowing the lawn;”
- The ability to turn on/off high data-generating mission-specific payloads and to archive this data.

Given these anticipated demands, we have integrated several specific ‘builds’ of the distributed command and power control system to implement these functions. In doing so, multiple gateways were configured, each with prototypical components attached in order to demonstrate the requisite functionality. These builds have been established with varying levels of fidelity to balance rapid development with realistic exercising of the system.

In performing these mission-oriented tasks we’ve been able to evaluate and iterate the design of the distributed computing and power control system to ensure that it can:

- properly interface with typical equipment/components,
- perform appropriate levels of local power and functional control (to include local PID and state machine controllers),
- perform parameter logging;
- support inter-gateway communication in a sufficient manner,
- log high-bandwidth streaming mission payload data without preventing the flow of lower-data rate but critical mission/vehicle control communications between gateways,
- respond to either pilot-based or automated navigation commands, and

- that it can provide arbitrary diagnostic information on system components.

Integrated system testing of this nature occurred at three different levels. Day-to-day 'bench test' testing was routinely run during formative design periods and significant demonstrations were conducted at each major design milestone and prior to elevating the testing to the next level. The second tier of testing mobile dry testing in which components were packaged in the small demonstration AUV mounted in a card, and the AUV was operated with team members providing locomotion forces in emulated surface operations (so, the cart was translated based on thruster operation and turned based on rudder angle; depth was not emulated). The third tier was operating the same AUV in the water first in a limited manner in MBARI test tank and soon in planned open water demonstrations in San Francisco Bay and/or local reservoirs.

8.1 Bench Test Demonstrations

The bench testing scenario was conducted by having all equipment laid out on a flat benchtop with the CAN and power buses running along the entire table. At various locations along the table, a gateway would be connected to the buses and components appropriate for the function of each gateway would be connected. Figure 8-1 shows the bench test equipment in typical layout configuration.

At various times, elements of this testing included networked gateways that performed the following functions:

- A mission control gateway that would determine sequences of commands to send to other gateways in order to execute mission-type activities;
- A mission payload gateway that controlled a high bandwidth 'instrument,' in this case a streaming webcam, that would stream and record its video to the mass storage drive;
- A pitch/depth control gateway that performed simulated closed-loop pitch/depth control using a stepper motor-driven mass slider, and IMU and a simulated depth sensor;
- A heading gateway that performed simulated heading control using an IMU/compass and the rudder;
- A GPS gateway to support surface-based positioning updates;
- A communication gateway to allow short range wi-fi based wireless communication with the vehicle for diagnostic and joystick-based piloting;
- A battery management gateway to allow intelligent control/monitoring of the power storage system;
- A mass drop system to drop ballast mass in case of an emergency;
- And other test equipment depending on specific demonstration objectives.

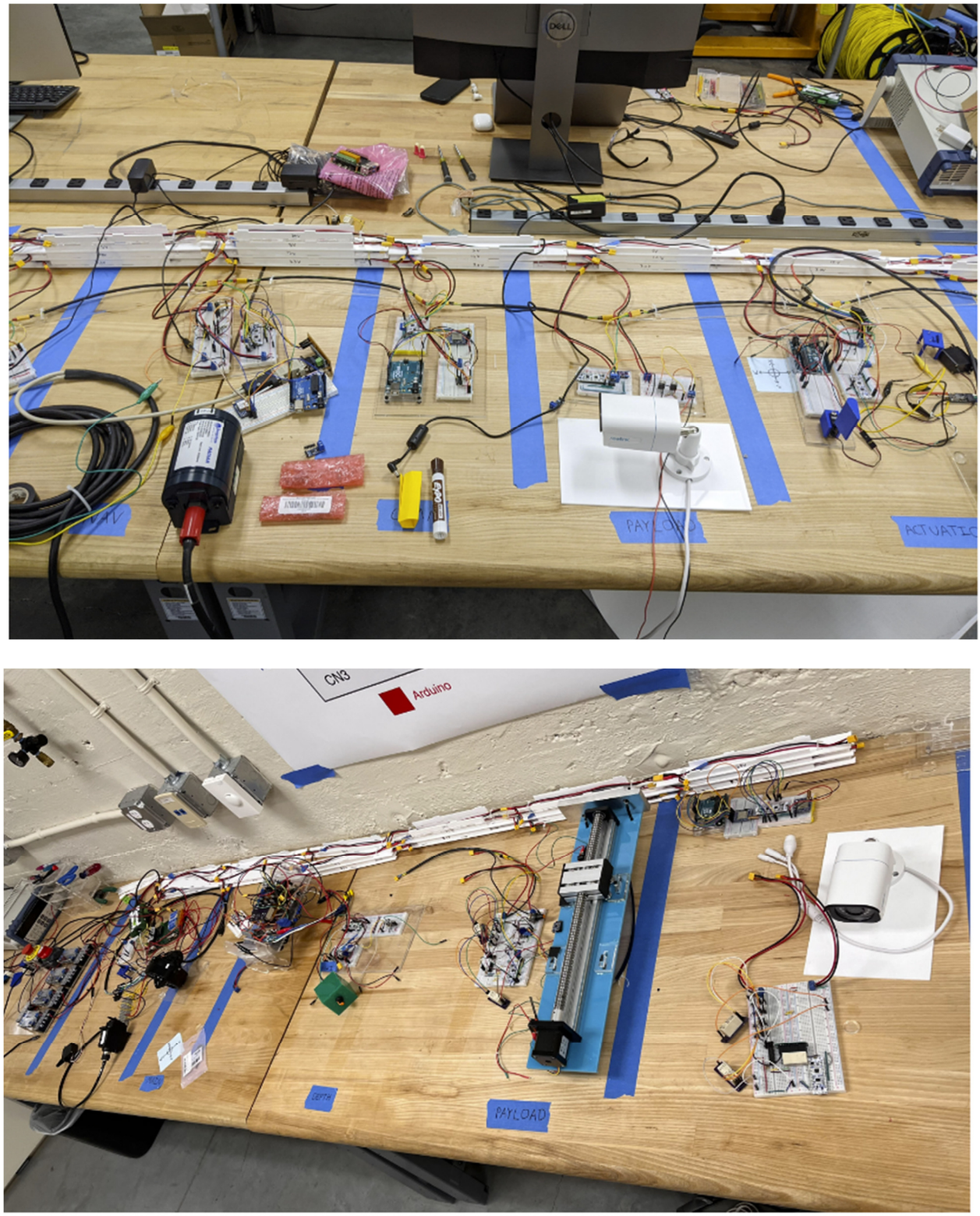


Figure 8-1: *Typical Gateway/Equipment Layout During Bench Testing*

This system was used for milestone demonstrations of mission-like functions. For example, in a simulated mission, the following demonstrations would be typical:

1. wirelessly run a brief diagnostic check of all subsystems (which might be done immediately before and after placing the AUV in the water),
2. show joystick control (which might be used to pilot the AUV away from a dock),
3. show heading and pitch/depth control (used for various navigation objectives) to include specific demonstrations of modes such as arbitrary diving, heading lock at a given depth, mowing the lawn maneuvers using heading control and timed leg translation, etc.
4. show high bandwidth payload data streaming and storage,
5. demonstrate data logging,
6. and so on….

We note that GPS-based waypoint and path control were not routinely demonstrated at this level given the indoor location of the test bench system. GPS data could be obtained to ensure it's operation, but the antenna that supported this was fixed, thereby not permitting easy emulation of a moving vehicle in terms of obtaining changing GPS data.

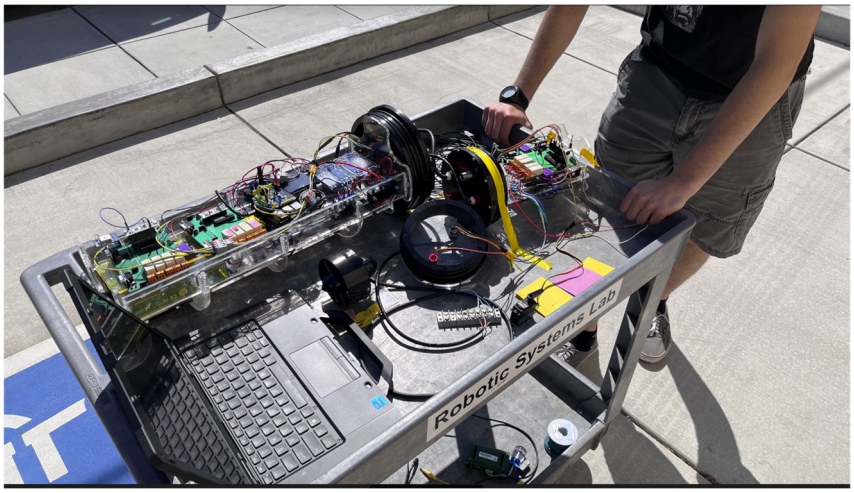
8.2 Simulated Surface Deployment with Vehicle on a Cart

In this test mode, a simple mission was conceived with necessary parts physically packaged in the simple test AUV. This required more realistic packaging, wiring, connections, etc. and time was spent to make these system elements more robust. It also put the system in a deployment-like physical arrangement where issues such as cable runs, interference, noise, etc. all came into play. As these tests were prepared for, all of these issues were addressed accordingly. Figure 8-2 shows an example of one of these types of tests.

For testing of this type that occurred in Spring 2023, the system was run in mission-like scenarios similar to the bench test scenarios. However, given the outdoor and mobile nature of the test, real GPS data could be used to monitor the change in position, the AUV could be moved to simulate motion based on thruster and rudder state, and realistic distances could be evaluated for issues such as wireless communication range. Given this, as an example, one of the mission emulations included the following tasks:

1. wirelessly run a brief diagnostic check of all subsystems (which might be done immediately before and after placing the AUV in the water),
2. show joystick control (which might be used to pilot the AUV away from a dock),

a) Simulating mobility by moving a cart based on thruster and rudder state



b) The test AUV equipment mounted in the AUV internal bottles

Figure 8-2: *Outdoor, mobile dry testing*

3. show GPS waypoint control to have the AUV move away from the dock to a defined 'mission start' location,
4. show surface (GPS) based navigation modes that included:
    a. heading control,
    b. "go to" waypoint control (which has an inner loop heading controller),
    c. path following (which also has an inner loop heading controller, but which is a more sophisticated navigation mode than simple waypoint control) to demonstrate this capability on the surface and how it will provide improved underwater navigation when a suitable underwater x-y positioning system is provided to the team,
    d. Mowing the lawn using a heading control and timed leg translation, etc.
5. show local area wireless communication beyond the range of a room (which was the case for bench testing),
6. show use of a deadman control link for safety purposes (e.g., the thruster stops if the deadman link is lost during surface test operations),
7. and so on….

We note that for these tests, pitch/depth control components and modes were not exercised given the limited nature of the AUV's mobility and the limited scope of the test.

8.3 Current Work on Open-Water Surface Field Deployment

This test – not part of the original project plan - is currently being planned for early Summer 2023 (beyond the term of the grant) as part of an unfunded student project. The test will be an open water surface test similar in nature to the cart-based test described in the previous subsection. That said, water-based testing will extend the fidelity of the testing environment in a manner that will certainly raise new challenges and issues to consider for the future.

The test will use the simple small-scale AUV system that was developed earlier in the project by a team of undergraduate students working for credit as part of a senior capstone project. Included components and their layout in the different AUV chambers are depicted in Figure 8-3. Preparation of the system is shown in Figure 8-4. This photo, taken during preliminary water testing in the MBARI saltwater test tank. In this test, remote wireless piloting/commanding was shown, thruster and rudder operation were demonstrated, and limited heading control was achieved (given the very limited space in which it was possible to demonstrate this function).

The system has been retrofitted with the distributed computing and power control system and will include the following components:

- Mission control gateway;
- WiFi connection via the wireless gateway;
- Heading control gateway to operate the thruster and rudder, to access heading data from an IMU, and to perform either pilot-directed thruster/rudder operations or to respond to vehicle commands for thrust and heading setpoints;
- GPS gateway to acquire position data;
- A small battery stack with battery monitoring gateway;
- The ballast-drop system to surface the vehicle in case of emergency.

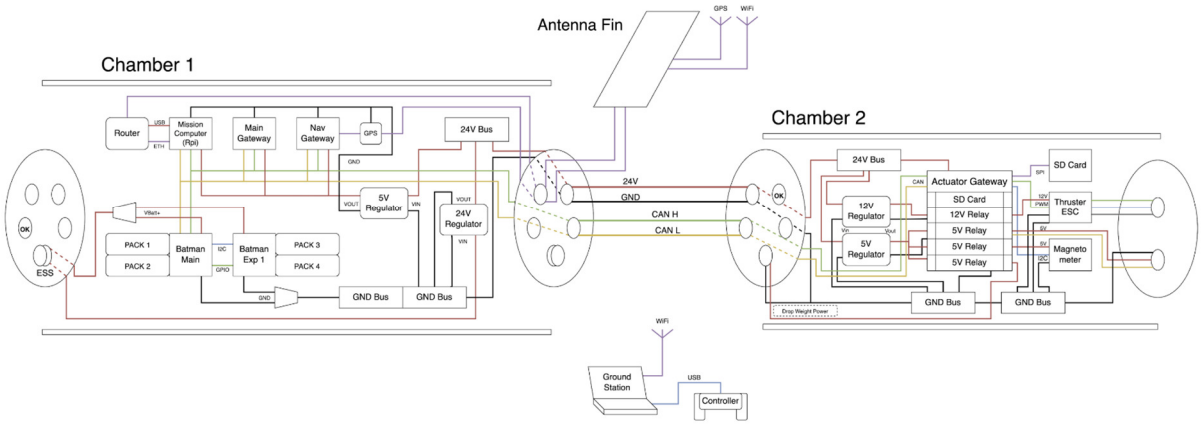


Figure 8-3: *Small AUV Prototype in the MBARI Saltwater Test Tank*

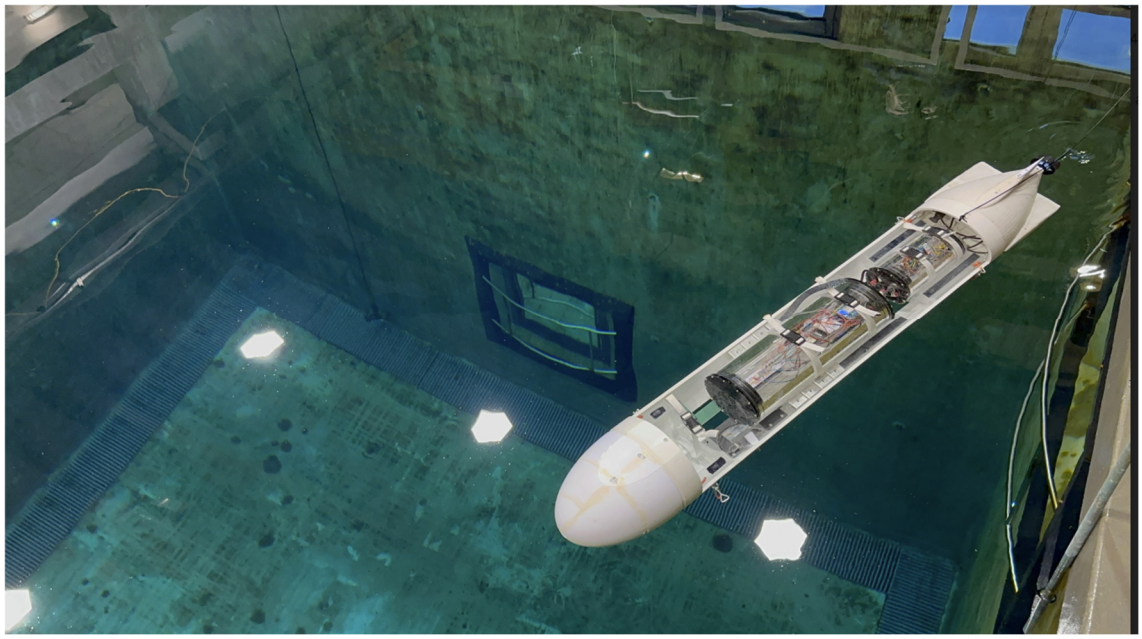


Figure 8-4: *Small AUV Prototype in the MBARI Saltwater Test Tank in Being Tested in Preparation for New Open-Water Tests*

Depending on logistics and approvals, the tests will most likely be performed from the shore at Stevens Creek Reservoir and/or in the Bay at the Port of Redwood City. A student crew has been trained on safe marine robotic procedures through an SCU course on that topic, and operating procedures for the test deployment are being finalized.

9.0 Summary of Work

The objective of this project was to explore the Huracan long-duration vehicle concept and to specifically develop a low-power distributed computing and power control architecture appropriate for this future class of AUV. Without question, Covid led to significant delays in conducting work, acquiring critical parts, getting access to design tools and facilities, etc. Nevertheless, we have designed a robust and capable system and have implemented several iterations of functional computing prototypes. We have also done extensive work to exercise and iterate the design of the system in the context of mission-oriented simulations and exercises ranging from bench testing to outdoor tests in a small test AUV prototype package.

Specific accomplished tasks included

- Definitions of Huracan-class AUV vehicle and mission concept;
- Definition of a distributed computing architecture capable of high resolution power configuration, local functional control, low-power implementation, and implementation in a physical implementation supporting component distribution over tens of feet;
- Design of a distributed computing interface specification and software libraries to support data communications between computing nodes for command, telemetry, and power control services;
- Design of a new computing node motherboard with local high resolution power configuration, the ability to run local real time functional controllers (for tasks such as heading control, instrument control, etc.), a variety of interface support for external components/peripherals, and the ability to be easily adapted to tailor the number of interface channels and to incorporate new interface specifications;
- Implementation of two revisions of a specific motherboard build for a set of candidate peripheral components and functions;
- Design of power system using a state-of-the-art high energy density battery cell and a custom control interface for power control and integration;
- Development of a bench model implementation of an entire integrated AUV electronics system using the developed distributed computing and power subsystems, simple candidate components for navigation (thruster, rudder servo, mass slide system for pitch control, heading and pitch sensor, GPS, etc.), communications (wireless surface comms), candidate instruments (streaming camera) with a high data rate storage bus/drive, remote surface driving and diagnostic capability, a safety mass drop device, a dead-man switch for testing, and a simple on-board mission scripting capability;
- Demonstration of the system in the context of a simple mission, packaged within a small demonstrator-model AUV shell, and commanded to conduct several mission-like functions such as heading control, navigating a planar mow-the-lawn pattern,

conducting a waypoint-based go-to function appropriate for surface navigation, etc.; these functions have been successfully demonstrated in the AUV as through simulated propulsion in an outdoor environment and are being prepared for an open water test in early June 2023.

The original vision of the Huracan initiative was to use this first Phase 1 effort to develop the distributed computing and power control system. The stated interest was that the program might continue with additional phases to explore other main subsystems/challenges inherent in the Huracan concept to include refinement and implementation of a complete power system, development of a low-drag hull, development of the guidance and control system (given challenges involving the length of the vehicle and the desire for low-drag actuation), etc. The SCU/MBARI team remain interested in exploring these follow-on developmental phases with the ultimate objective of producing one or more real, functional Huracan AUVs for field use.

Archive & References

This final report summarizes the design work performed in this phase of the Huracan program. Significant additional documentation exists as part of a design archive at SCU, which includes schematics, board layout files, software source code, internal design reports and presentations, etc.

Some useful, mainly external references used in the project include:

- The MBARI LRAUV design archive
- The SCU Huracan design archive
- Related SCU project reports:
    - Z. Cameron, “Distributed Computing and Power Control Architecture for a Long Range AUV,” Capstone Report, MSEM&L, Santa Clara University, Adv: C. Kitts, June 2021.
    - R. Konrath, E. Kennar, J. Taylor, A. Knaus, S. Kidd Meyers, D. Guitierrez, “The Modular Oceanic Autonomous Underwater Vehicle for Novel Actuation (MOANA) 2.0,” Senior Capstone Report, Adv. C. Kitts, June 2022.
    - Z. Karat, “MOANA Navigation and Functional Demonstrations Using a Low Power Distributed Computing and Power Control System,” DRAFT, MS Capstone Report, Advs. M. Neumann and C. Kitts, June 2023 expected.
- Selected references on AUVs
    - “Autonomous underwater vehicles,” May 2018. [Online]. Available: https://www.mbari.org/at-sea/vehicles/autonomous-underwater-vehicles/
    - B. Hobson, J. Bellingham, B. Kieft, R. McEwen, M. Godin, and Y. Zhang, “Tethys-class long range auvs - extending the endurance of propeller-driven cruising auvs from days to weeks,” 09 2012, pp. 1–8.
    - “Long-range autonomous underwater vehicle tethys,” Aug 2018. [Online]. Available: https: //www.mbari.org/at-sea/vehicles/autonomous-underwater-vehicles/long-range-auv-tethys/
- Selected references on distributed computing systems:
    - L. L. Bello, R. Mariani, S. Mubeen, and S. Saponara, “Recent advances and trends in onboard embedded and networked automotive systems,” IEEE Transactions on Industrial Informatics, vol. 15, no. 2, pp. 1038–1051, 2019.
    - B. Palmintier, “The emerald protocol suite: Design and implementation of a modular distributed architecture for small satellite command, telemetry, and power systems,” Engineer, Stanford, 2004.
    - M. Swartwout, C. Kitts, P. Stang, and E. G. Lightsey, “A standardized, distributed computing architecture: Results from three universities,” in 19th Annual AIAA/USU Conference on Small Satellites, 2005.
    - S. Mu˜noz, J. Greenbaum, G. Lightsey, T. Campbell, S. Stewart, and G. Holt, “The fastrac mission: Operations summary and preliminary experiment results,” 2011.

    - F. Rogers-Marcovitz and P. Williams, “Deep: Dallas eeprom equipment profile for rapid integration and automatic system modeling,” 2007.
- Selected references on computing options: CAN, microcontrollers, etc.
    - ”Introduction to the Controller Area Network (CAN)”, ”Texas Instruments”, 4 2016.
    - J. Grith, “What do can bus signals look like?” [Online]. Available: https: //e2e.ti.com/blogs /b/industrialstrength/posts/what-do-can-bus-signals-look-like
    - SN65HVD23x3.3-V CAN Bus Transceivers, Texas Instruments, 4 2018.
    - W. Lun, C. K. Ng, B. Mohd Ali, N. Ali, F. Noordin, and F. Rokhani, “Review of researches in controller area networks evolution and applications,” Proceedings of the Asia-Pacific Advanced Network, vol. 30, 08 2010.
    - “Buy a raspberry pi zero.” [Online]. Available: https://www.raspberrypi.org/products/raspberry-pi-zero/
    - “Arduino uno rev3.” [Online]. Available: https://store.arduino.cc/usa/arduino-uno-rev3
    - ”Ultra-low-power Arm® Cortex®-M4 32-bit MCU+FPU, 100DMIPS, up to 256KB Flash, 64KB SRAM, USB FS, analog, audio”, ”ST”.
    - “Why a real-time operating system?” [Online]. Available: https://micro.ros.org/docs/concepts/rtos/
    - “Teensy® usb development board.” [Online]. Available: https://www.pjrc.com/teensy/
    - “Cortex-m0+.” [Online]. Available: https://developer.arm.com/ip-products/processors/cortex-m/cortex-m0-plus
    - Kinetis KL26 Sub-Family, Freescale Semiconductors, Inc, 8 2014, rev. 5.
    - “Cortex-m4.” [Online]. Available: https://developer.arm.com/ip-products/processors/cortex-m/cortex-m4
    - Kinetis K66 Sub-Family, Freescale Semiconductors, Inc, 4 2017, rev. 4.
    - “Cortex-m7.” [Online]. Available: https://developer.arm.com/ip-products/processors/cortex-m/cortex-m7
    - i.MX RT1060 Crossover Processors for Consumer Products, NXP Semiconductors, 3 2021, rev. 2.
    - du2013, “du2013/snooze,” Oct 2020. [Online]. Available: https://github.com/du2013/Snooze
    - “Nucleo-l432kc.” [Online]. Available: https://os.mbed.com/platforms/ST-Nucleo-L432KC/
    - F. Pastors, “Implementation of canopen and canopen fd.” [Online]. Available: https: //www.canopensolutions.com/english/aboutcanopen/implementation.shtml
    - “Sn65hvd230 can board.” [Online]. Available: https://www.waveshare.com/sn65hvd230-can-board.htm
- Selected references on AUV dynamics:
    - E. Almendinger, Submersible Vehicle Systems Design, The Society of Naval Architects and Marine Engineers, 1990.

    - T. Prestero, “Verification of a six-degree of freedom simulation model for the remus autonomous underwater vehicle,” Massachusetts Institute of Technology, Sep. 2001. [Online]. Available: https://core.ac.uk/download/pdf/4429735.pdf.
- Selected references on proxy AUV components for testbed prototyping:
    - “Bmp280 barometer.” [Online]. Available: https://www.makerfabs.com/bmp280-barometer.html
    - “Adafruit ultimate gps breakout - 66 channel w/10 hz updates.” [Online]. Available https://www.adafruit.com/product/74658
    - “Servo motor sg-90 basics, pinout, wire description, datasheet.” [Online]. Available: https://components101.com/motors/servo-motor-basics-pinout-datasheet
    - “Microsd card breakout board+.” [Online]. Available: https://www.adafruit.com/product/254
    - “Adafruit rfm95w lora radio transceiver breakout.” [Online]. Available: https://www.adafruit.com/product/3072

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing the burden, to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.
PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
05/19/2023	Final	4/1/20 - 4/19/23

4. TITLE AND SUBTITLE
HURACAN: 21" Long Range Autonomous Underwater Vehicle

5a. CONTRACT NUMBER
N00014-20-1-2324

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Dr. Christopher Kitts

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Robotic Systems Laboratory, Santa Clara University, 500 El Camino Real, Santa Clara CA 95053
Monterey Bay Aquarium Research Institute, 7700 Sandholt Rd. Moss Landing, CA 95039

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Office of Naval Research, 875 N. Randolph Street, Suite 1425, Arlington, VA, 22203-1995 (sponsoring agency).
ONR REG Office Seattle, Telephone 206-548-7234,Email:ONR_Seattle@navy.mil, 300 Fifth Avenue, Suite 710, SEATTLE WA 98104-3251 (administrative office)

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Unclassified Unlimited

13. SUPPLEMENTARY NOTES
Ref Proposal Number N4992NVONR;
Grant POC Email: vanessa.seymour@navy.mil; Technical POC: daniel.deitz@navy.mil

14. ABSTRACT
This project explored the development of a new Autonomous Underwater Vehicle (AUV) reference design, known as Huracan, capable of very long duration autonomous operations on the order of weeks or possibly more than a month. This Phase 1 project focused on the design and prototyping of a new, distributed, low-power computing and power control system given the critical role that such a system would play in any feasible design. Such a design was successfully achieved with the definition of a computing architecture, development of new customer low-power distributed computing gateways, and evaluation/iteration of the design through several levels of system prototyping.

15. SUBJECT TERMS
Autonomous Underwater Vehicle (AUV)
Distributed computing and power control

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Dr. Christopher Kitts
U	SAR	SAR	None	73	19b. TELEPHONE NUMBER *(Include area code)* (408) 554-4382