

JaiaBot: Open Source Multi-AUV System for Scalable Data Acquisition

Toby Schneider

GobySoft, LLC

Falmouth, MA, USA

toby@gobysoft.org

Jason Webster

JaiaRobotics, Inc.

Bristol, RI, USA

jason@jaia.tech

Ian Estaphan Owen

JaiaRobotics, Inc.

Bristol, RI, USA

ian@jaia.tech

Raymond Catania

Catania Product Development

Melbourne FL, USA

rcatania@cataniaenterprises.com

Matthew Ferro

JaiaRobotics, Inc.

Bristol, RI, USA

matt.ferro@jaia.tech

Michael Rock

JaiaRobotics, Inc.

Bristol, RI, USA

michael.rock@jaia.tech

Abstract—The JaiaBot system leverages open source software and modular electronics combined with innovative hydrodynamics to produce a low cost multi-vehicle fleet for shallow water data collection. By utilizing a design that is a hybrid of a surface vehicle and a profiling float, the JaiaBot can perform point measurements in depth while avoiding expensive underwater navigation and communications components. Applications for the JaiaBot have been wide-ranging due to its flexible payload capabilities and ease of integration due to its open source software design.

Index Terms—autonomous underwater vehicles, multi-robot systems, automated data collection

I. INTRODUCTION

A JaiaBot fleet consists of a variable number (as many as 15 have been fielded at one time) of small AUVs (the Bots) and a shore- or boat-based base station (the Hub). The Bots weigh 3 kg, are 96 cm in length and 7 cm in diameter, and run at speeds up to 5 m/s. They perform surface transits to the desired waypoints at which point they perform a task: a profiling dive (up to 60 meters deep), a surface drift for a specified duration to capture currents, or an active station-keep for single-point long duration data collection of wave statistics.

The innovations of the JaiaBot system are substantially enabled by the fully open source software system running on all the Bots and Hubs. By adopting open source software, the JaiaBot system has a head start on the core vehicle software subsystems, and thus new innovation is targeted at application and specific data collection goals. Some example goals include wave height estimation, sea bottom type classification, and water column profiling (e.g., salinity, temperature, sound speed).

The vehicle hydrodynamics are innovative allowing high speed transits. As the Bots are passively vertical at rest they easily perform profiling dives (in reverse, see Fig. 1). The modular electronics system is largely based on readily available commercial products such as the Raspberry Pi ARM64 CPU, and the Arduino Nano.

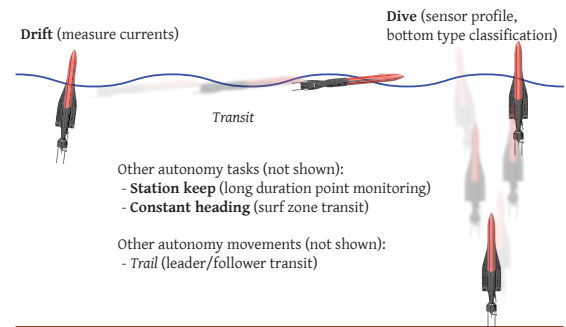


Fig. 1. Visual depiction of the JaiaBot Autonomous Tasks and Movements. A mission can be composed of any number of Movements (e.g., Transit) followed by any Task (e.g., Dive, Station Keep) to fulfill the data collection goals of the particular application (See Fig. 3 for the various Task and Movement states). Dives can either hit the bottom or a desired target depth. This figure shows an example Drift task followed by a Transit movement, followed by a Dive task.

A. Related Work

Most existing AUV platforms broadly fall into two categories: 1) AUVs (both active propulsion and glider-types), which transit primarily underwater (and thus require underwater navigation and communications), 2) autonomous surface vehicles (ASVs), which do not transit underwater and thus can only perform remote sensing below the near surface zone. If one considers them to be AUVs, a third category would be profiling floats (such as ALACE [1], ARGO [2], and SOLO).

Based on this categorization, the JaiaBot system is essentially a fleet of miniature ASVs with near shore profiling float capability. This allows the cost of the system to be dramatically reduced compared to underwater-transit AUVs (primarily by removing the need for acoustic-based navigation and communication), but the system is still able to do direct measurement of underwater properties by low-cost, low-power direct measurement sensors (e.g., a CTD) or remote sensing equipment (e.g., a hydrophone). On the other hand, unlike a traditional profiling float, the JaiaBot system has full autonomous control to perform profiles (and other tasks)

precisely where the operator needs data, rather than being dependent on the free floating Lagrangian nature of traditional profiling floats.

Other relevant existing hybrid AUV systems include 1) the Folaga AUV [3] and its successor the X-300, which is an ASV/glider hybrid and 2) the Aquabotix SwarmDiver (a now-defunct commercial AUV), and 3) the Diving Autonomous Surface Vehicle (DASV) [4], which has a similar concept of operation but uses an active variable buoyancy system to dive.

II. SOFTWARE

The utility and functionality of modern robotic systems are substantially driven by their software. The range of tasks required by AUV software, especially multi-vehicle systems, are complex and span a number of domains: control, autonomy, communications (internal and external), data management, health monitoring, user interfaces, mission planning, and more.

Many of these problems have solutions that already exist in the open source community. For this reason, the JaiaBot software (see Fig. 2) is built on several existing such projects, primarily Goby3 (for communications) [5] and MOOS-IvP (for autonomy) [6], and is itself released fully open source¹.

The JaiaBot software system has been successful largely due to two approaches: 1) substantially leveraging high-quality open source software projects and 2) management of complexity along the various orthogonal axes of software development (see Section II-F).

A. Autonomy and Control

The JaiaBot autonomy system is comprised of the existing C++ MOOS-IvP behavior based autonomy (which provides a collection of vehicle heading and speed behaviors such as waypoint traversal and station keeping in currents) and a C++ class-based state machine based on the open source Boost Statechart project [7]. The state machine (shown in Fig. 3) governs the entire vehicle lifecycle from power-on (“PreDeployment”) through deployment and in-water operations (“InMission”) to post-mission data offload and shutdown (“PostDeployment”).

This hybrid state-machine/behavior-based design provides the rigour and predictability of a state machine with the flexibility of the real-time multi-objective optimization from the IvP Helm (behaviors shown in yellow in Fig. 3). A certain degree of predictability is important to AUV operators, along with robust system design, which minimizes behavior edge cases. At the same time within a particular state the JaiaBot can support reactive autonomy by use of the MOOS-IvP behavior based framework.

The PID-based control system is run on an Arduino micro-controller for realtime performance and ease of programming, and communicates via a serial link to the autonomy system to constantly update control setpoints.

B. Communications

The communications system is based on the Goby3 nested autonomy communications, which provides a common publish/subscribe API to three different levels within the system: interthread (within a process), interprocess (within a Bot or Hub), and intervehicle (between the Bots and the Hub).

This approach allows the system to manage the inherently different data throughputs available between each layer (e.g., the intervehicle XBee radio provides dramatically less throughput than the on-board Ethernet) while at the same time providing a unified mental model for the programmer through the common API and similar message formats (primarily Google Protocol Buffers and DCCL).

C. Data Management

During the mission, some data such as vehicle position, health, and dive data profiles are sent over the radio link and visualized on the map-based Jaia Command & Control (JCC) user interface (written in Python and TypeScript).

Data are logged on-board each Bot using an open binary format (.goby) which is offloaded by the operator to the hub via Wi-Fi after the mission. On the Hub the .goby files are converted to Hierarchical Data Format version 5 (HDF5), a widely supported structured scientific data format that can be read from all major programming languages (e.g., MATLAB, Python, Julia, C++, etc.). Additionally, the Jaia Data Vision user interface can provide easy visualization of the vehicle data streams immediately after the mission.

D. User Interfaces

All the user interfaces are provided as web servers running on the Hub, so the user can connect with any browser available on a tablet or laptop computer. This includes 1) the main user interface, Jaia Command & Control (JCC), which is a TypeScript/Python (Flask) web application, 2) the data visualization tool, Jaia Data Vision (JDV), which is a Python web application, and 3) the offline upgrade UI, Jaia Configuration & Upgrade (JCU), based on Goby Liaison (from the Goby3 project). See Fig. 9 for an example of JDV.

E. Cloud-based Digital Twin

A virtual JaiaBot system (a “Digital Twin”) can be run for any fleet in the Amazon Web Services (AWS) cloud. It is an identical replica of the software running on the real system except the CPU architecture is different, sensors and actuators are simulated using their equivalent real data input and the virtual system can be run faster than realtime. Each Bot and Hub in the fleet is represented by its own virtual machine, with a near replica of the filesystem run on the Raspberry Pi computers in the field.

This Digital Twin provides an excellent training, development, and pre-mission test environment at a much lower cost and difficulty than fielding vehicles. Additionally, it allows operator access for command and control of real fleets from any internet connected location.

¹LGPL/GPL licensed: <https://github.com/jaiarobotics/jaiabot>

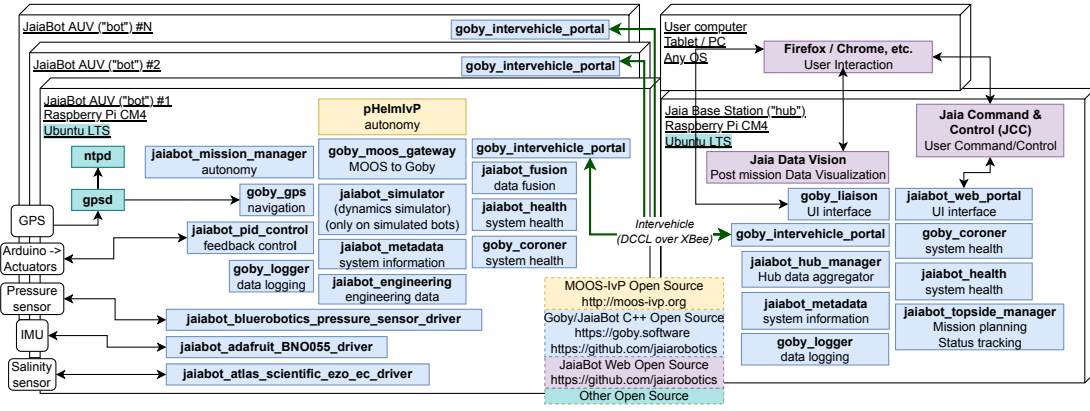


Fig. 2. JaiaBot Software structure diagram showing the applications running on the bots and hub and the relevant open source project. Communication links between the sensors and associated drivers are shown, and all the other applications communicate through the applicable publish/subscribe middleware (Goby3 [gobyd, not shown] or MOOS [MOOSDB, not shown]).

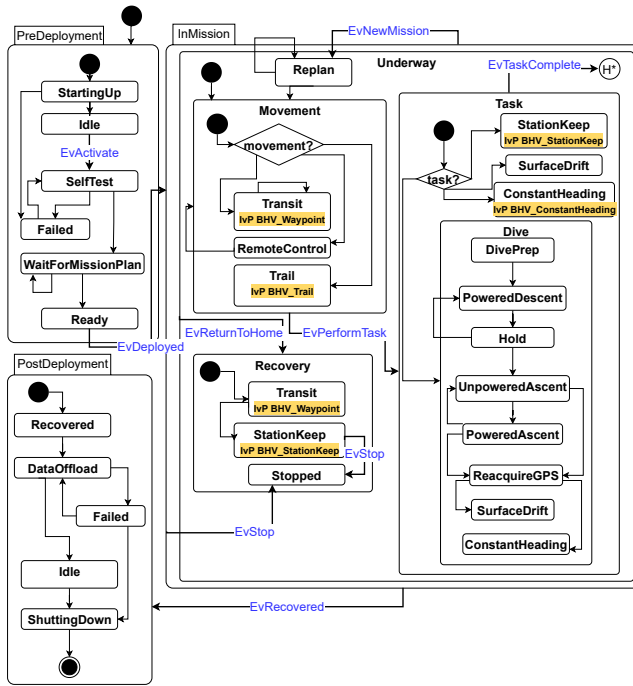


Fig. 3. JaiaBot State Diagram (for clarity, only key events are shown). All the states are implemented in C++ using Boost Statechart and some states oversee specific MOOS-IvP behaviors as shown. A typical mission progresses from PreDeployment → (Underway:Movement → Underway:Task) [repeat] → Underway:Recovery → Underway:PostDeployment

F. Complexity Management

Management of complexity is a critical part of developing successful capable multi-robot systems, and is thus an important aspect to the JaiaBot system development and maintenance. By their nature, robotic systems require integration of a wide range of different subsystems and without careful design this can lead to unmanageable complexity leading to unpredictability, failure, or inability to upgrade or add further capabilities.

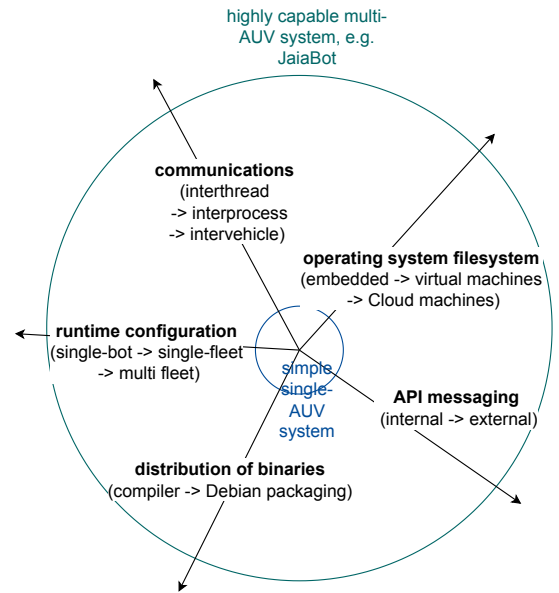


Fig. 4. Sketch representing several relevant axes of software components where complexity can grow as the capability and scale (number of vehicles) of the AUV system grows. The JaiaBot system manages this complexity largely by using a single point of reference (typically files in the jaiabot git repository) and a wide range of automated software tools that provide reproducible results.

A number of “axes” where complexity has a tendency to grow within the JaiaBot system are shown in Fig. 4. For each of these axes, the JaiaBot system has been able to stay manageable by a small team by utilizing a few key principles:

- All software byproducts are generated from an original source in the jaiabot git repository. This ensures that a change to the system in one place means a change everywhere. For example, the Ubuntu filesystem used for the embedded Raspberry Pi, VirtualBox virtual machine testing, and AWS cloud images are all automatically compiled from the same set of files. Similarly, the API messaging for both internal (Goby3 publish/subscribe)

and external (JSON REST API) are all derived directly from the same set of Google Protocol Buffers messages (using introspection to derive the JSON REST API).

- Automated tools are used to build all downstream outputs wherever possible, and check the validity of human changes within the system rules (“linting”). The most obvious use is the C++ compiler’s ability to check syntax, typing, etc., which we extend (using Goby3 tools) to provide physical unit safety (static dimensional analysis) and publish-subscribe directed graphs (showing which publisher produces a particular data type required by a subscriber). Other examples include the use of web-based “DevOps” tools (CircleCI) to produce binary Debian packages automatically from the source code. We have a linter that checks for key changes to intervehicle messaging to ensure (human) developers do not accidentally make backwards-incompatible changes.

III. MECHANICAL AND HYDRODYNAMICS

The JaiaBot has numerous mechanical design innovations including some with pending patents. See Fig. 5 for a mechanical diagram of the JaiaBot.

A. Surface Transits and Profile Dives

The concept of the JaiaBot transiting on the surface, but diving vertically for data collection allows for faster mission tempo, as the drag forces are much less for a surface vehicle than an underwater vehicle, resulting in higher transit speeds. Additionally, the monetary cost of navigation on the surface is much lower with low cost GPS units having the accuracies of underwater solutions costing orders of magnitude more. Finally, having near-constant comms with the vehicle (when in range or using cellular/satellite) allows for a much better user experience with real time data displays and retasking of multiple vehicles with the push of a button.

B. High Speed

An early design criteria for the JaiaBot was high speed surface transit to allow use in high current environments such as estuaries and rivers. To achieve this high speed, a streamlined central propulsor and planing wings were developed using CFD software.

C. Passive Dynamic Center of Mass

The JaiaBot has a passive center of mass adjustment system that helps it change orientation from horizontal to vertical and vice-versa. While vertical, the vehicle dives straight allowing for accurate vertical profiles. The center of buoyancy’s relation with the center of mass has been optimally separated to aid in the verticality of profile dives as well. These features allow the JaiaBot to eschew complex maneuvers such as spiraling or angling which are often necessary for other subsurface vehicles to dive.

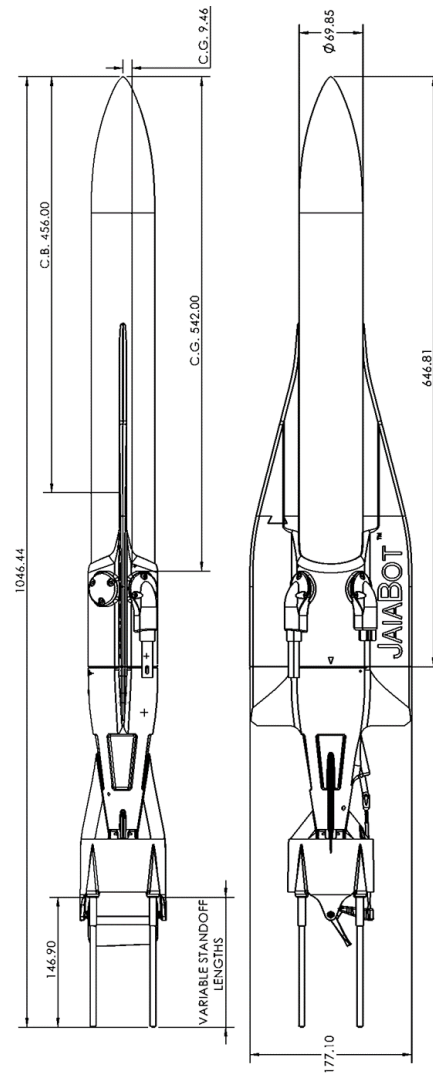


Fig. 5. Mechanical drawing of the JaiaBot indicating its center of buoyancy and center of gravity. All dimensions are in mm.

D. Trimming

Additionally, since the JaiaBot is largely positively bouyant, trimming the vehicle based on the salinity is unnecessary. This also allows small payloads to be added (or removed) without the user needing to spend time adjusting ballast. These time savings add up quickly with fleets of multiple vehicles.

E. Modularity

The underwater robotics world employs a host of different connectors which makes it difficult and expensive to provide modularity to users. To keep cost and complexity low, custom expansion ports were designed with simple face-sealing O-rings that allow 3D printed expansion pods to be mounted and swapped.

IV. ELECTRONICS

The electronics (see Fig. 6) for the JaiaBot have been selected for ease of implementation, cost, and availability.

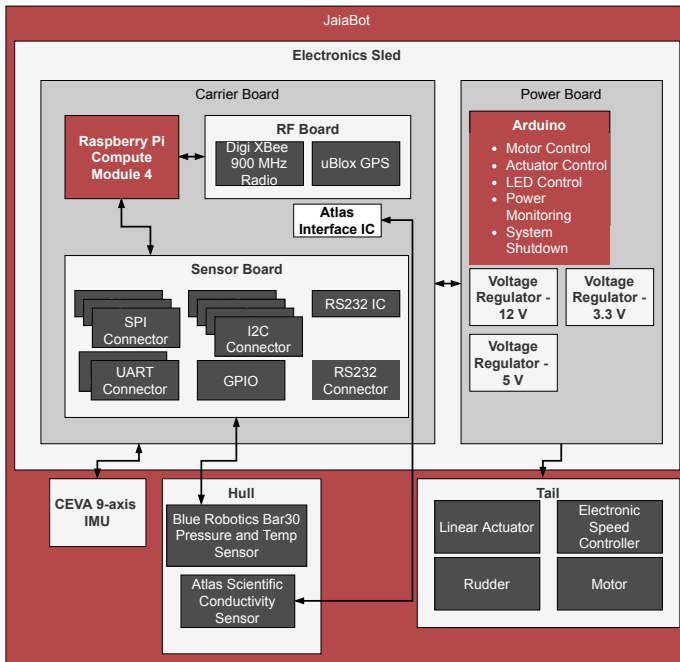


Fig. 6. High level block diagram of the electronics system in the core JaiaBot highlighting the modular design.

Each component in the stack from the Raspberry Pi to the Blue Robotics Bar30 pressure sensor has readily available documentation and libraries for support. They were also available during the height of COVID-19 supply chain issues, when the product was conceived and launched.

Modularity is at the heart of the electronics design allowing Jaia Robotics or its customers to easily change radios, GPS chips, IMUs, sensors, etc. with minimal rework internally. For instance, the radio and GPS are on their own daughter card which can be replaced with a modified design such as an RTK GPS or a cellular radio without changing any of the other vehicle electronics.

New sensors and other peripherals can be added with ease due to the breakout of many protocols from the Raspberry Pi Compute (including UART, SPI, I2C, and USB) and multiple voltage rails (3.3 V, 5 V, 12 V, and full battery). If new combinations of the aforementioned are required, a small \$40 PCB can be made available with very little design time.

V. EXISTING APPLICATIONS AND DATA COLLECTION

Some existing applications for the JaiaBot system are:

- acoustic recording, as showing in Fig. 7.
- surf zone observations to determine the relative safety of passage. An example data product using the Jaia Data Vision post-mission data analysis user interface is shown in Fig. 9. Within a single mission, the JaiaBot can measure bathymetry, currents, and estimated wave statistics using just the onboard GPS, IMU, and pressure sensor.
- identification and characterization of rip currents
- sound velocity profiling

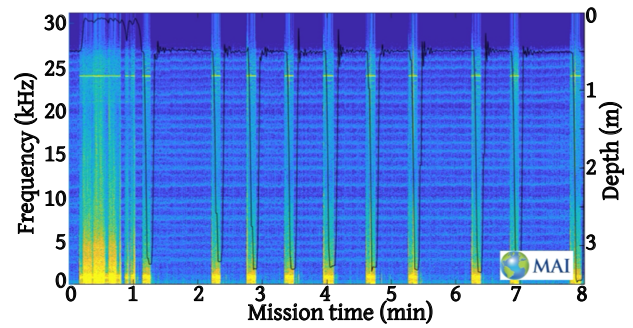


Fig. 7. Acoustic spectrogram of recordings from a JaiaBot-PAM while diving repeatedly (The vehicle depth is overlaid in black on the spectrogram.). The vehicle can record ambient data while floating to the surface as the propeller is off and thus self noise is low at this time.

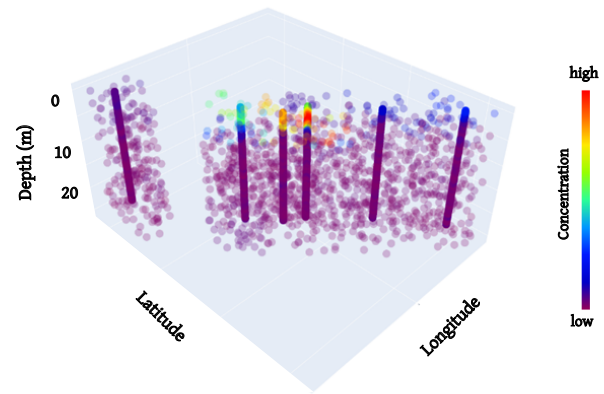


Fig. 8. Three dimensional fluorescein concentrations about a simulated oil spill measured from a JaiaBot-FLUOR. Solid dots are measurements, and translucent dots are interpolated data.

- investigation of aggregate data collection and analysis for increased fidelity over individual capabilities (see, for example, [8])

Active development efforts with partners are adding capabilities to

- collect eDNA samples remotely
- detect, localize, and characterize spills and contaminants. See Fig. 8 for a simulated oil spill data collection experiment.
- determine overall water quality and ascertain the presence of harmful algal blooms
- frontogenesis analysis
- acoustic modem integrations

REFERENCES

- [1] R. Davis, J. Sherman, and J. Dufour, "Profiling ALACEs and other advances in autonomous subsurface floats," *Journal of atmospheric and oceanic technology*, vol. 18, no. 6, pp. 982–993, 2001.
- [2] D. Roemmich, G. C. Johnson, S. Riser, R. Davis, J. Gilson, W. B. Owens, S. L. Garzoli, C. Schmid, and M. Ignaszewski, "The Argo program: Observing the global ocean with profiling floats," *Oceanography*, vol. 22, no. 2, pp. 34–43, 2009.
- [3] A. Alvarez, A. Caffaz, A. Caiti, G. Casalino, L. Gualdesi, A. Turetta, and R. Viviani, "Folaga: A low-cost autonomous underwater vehicle combining glider and AUV capabilities," *Ocean engineering*, vol. 36, no. 1, pp. 24–38, 2009.

