Coordinating Multiple Autonomies to Improve Mission Performance

Conlan Cesar
Dept. of Mechanical Engineering
Massachusetts Institute of Technology
Cambridge, Massachusetts, USA
conlanc@csail.mit.edu

Michael DeFilippo
Dept. of Mechanical Engineering
Massachusetts Institute of Technology
Cambridge, Massachusetts, USA
mikedef@mit.edu

Michael R. Benjamin
Dept. of Mechanical Engineering
Massachusetts Institute of Technology
Cambridge, Massachusetts, USA
mikerb@csail.mit.edu

Ben Whetton
SeeByte ltd.
Edinburgh, UK
ben.whetton@seebyte.com

Patrick Opgenoorth
SeeByte ltd.
Edinburgh, UK
patrick.opgenoorth@seebyte.com

Andrea Munafò
SeeByte ltd.
Edinburgh, UK
andrea.munafo@seebyte.com

***Abstract*—Research in marine autonomy has traditionally involved development around a single autonomy architecture. Even when an architecture is open and modular, modules are typically all integrated into a single architecture. In this paper, an approach is considered that couples two distinct and mature architectures into a single combined autonomy system that leverages the strengths of both singular architectures to produce a more complete and capable system. In this work, the two autonomy architectures, SeeByte's Neptune, and MIT's MOOS-IvP, are combined via a shared interface to leverage SeeByte's mission path-planning and exclusion zone capabilities along with the reactive path execution with obstacle and collision avoidance behaviors of MOOS-IvP. Results are reported from simulation and on-water tests held on the Charles River in Cambridge MA during July 2021. The Neptune/IvP combined system was deployed on MIT's autonomous Boston Whaler and used to perform a safe crossing in a busy and dynamic environment.**

***Index Terms*—autonomy, moos-ivp, neptune, moos, ros, marine robotics, interface, collaboration, decision making, path planning, multi-objective optimization**

1. Introduction

Autonomous systems are being routinely deployed in a number of scenarios [1], ranging from scientific and civil applications such as oceanographic exploration [2], [5] or infrastructure monitoring and surveying [6], to military operations where autonomous vehicles are used to carry out autonomous surveys and inspect for possible threats [3], [4].

Commercially deployed systems typically allow the operator to plan and execute a small number of autonomous operations, such as lawnmower survey patterns, but are limited in terms of adaptation or extendibility. In parallel, most research in autonomy has focused on developing new behaviors (to control the vehicle) or sensor processing algorithms (to improve the robots perception of its environment). This has resulted in researchers focusing their development around the use of a single autonomy architecture with little consideration on how best to leverage capabilities from multiple architectures to produce the most complete and capable system.

Our work aims at bridging this gap to allow capabilities integrated into multiple autonomy architectures to be leveraged concurrently based on mission requirements and for those architectures to cohabit on a single platform.

The paper will describe how two autonomy architectures, namely SeeByte's Neptune [10] and MIT's MOOS-IvP [11] have been integrated together to balance variable levels of autonomy coordinating behaviors residing on different systems using a pass-through architecture (Fig. 1). This architecture is based on hierarchical delegation of authority. The integrated system leverages Neptune's high-level mission planning based on asset capabilities and exclusion zones to provide objectives to MOOS-IvP. MOOS-IvP translates these plans into waypoints and executes them, making available to the vehicle its powerful and flexible reactive planning.

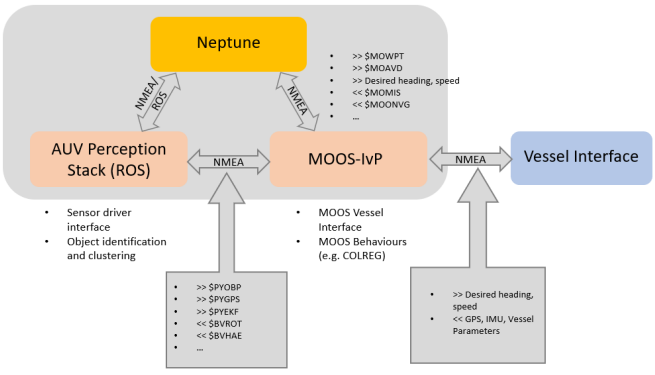

Figure 1. Conceptual representation of the interactions among autonomies

Results are reported through simulations and in-water

deployments in a scenario where an autonomous vehicle is tasked to calculate a safe path to cross a busy river, minimising mission time while avoiding obstacles and collisions. The vehicle is equipped with camera, lidar, and radar sensors to detect obstacles in real-time. In this scenario, Neptune tackles higher-level planning and provides a feasible route to avoid the main clusters of obstacles, replanning as the situation evolves. MOOS-IvP receives the higher-level tasking and executes it, while also applying the Avoidance Behavior to reactively avoid obstacles. The system has been deployed on MIT's ASV (Autonomous Surface Vessel) RoboWhaler, which is the autonomous vessel built upon the research vessel Philos (Fig. 2). Experimental results are presented from in-water tests performed in the Charles River in July 2021.

The rest of the paper is organised as follows: Section 2 describes SeeByte's Neptune autonomy and the higher level planning. Section 3 describes the MOOS-IvP architecture and the obstacle avoidance behaviour. Section 4 describes possible solutions to integrate multiple autonomies together onto a single robot. Section 5 goes into the details on how Neptune and MOOS-IvP have been integrated together, and hence able to negotiate responsibilities based on the mission evolution. Section 6 describe simulative and experimental results. Finally, conclusions are drawn in 7.

Figure 2. R/V Philos (RoboWhaler). RoboWhaler is an autonomous surface vessel that is used for advanced autonomy and sensor fusion algorithm development.

2. SeeByte Neptune

Neptune is a decentralised goal-based planner and autonomy engine that is able to optimise and adaptively plan the execution of a mission (see Fig. 3). It acts at the command and control level to produce optimised offline multi-vehicle plans and at vehicle level to automate the mission execution using a modular, and open behaviour-based architecture. Neptune's decentralised autonomy architecture makes it possible to break down the mission goal into multiple tasks that can be optimally allocated to the all the vehicles in the fleet. Multiple tasks are run in parallel and the available vehicles automatically take responsibility for tasks. This acts as a force multiplier as each vehicle is able to benefit from all the information of every other vehicle in the fleet: they are able to cover far greater areas at a time and the operation is not affected if a vehicle malfunctions.

Neptune relies on the OODA (Observe, Orient, Decide, Act) paradigm, and is composed of two key elements: Neptune Topside, which is a mission planner and monitor (Neptune Command and Control - C2), and Neptune Wetside, which runs on the vehicles and is responsible for mission execution and vehicle-to-vehicle coordination. To maximise modularity and flexibility, Neptune Wetside is composed of four main elements:

- Functions to monitor the internal state of the vehicle and its environment;
- Core Autonomy modules to provide an understanding of the context of the mission such as the surrounding world and the no-go areas. One key part of the core Autonomy modules is the Neptune Arbiter, which is responsible for deciding the order of the steps in the mission and for delegating the execution to the relevant behaviours.
- Neptune Behaviors to enable the robot to move in the world and fulfil its mission.
- Finally, to ease its integration with the robots/sensors, Neptune provides dedicated API and functionalities provided through SOLAR, a commercially controlled open API.

An overview of Neptune high-level architecture is provided in Fig. 3.

It is also worth highlighting that providing a shared autonomy architecture, Neptune natively supports acoustic communications and all Neptune-enabled vehicles are able to communicate with each other through a shared 'language'. This effectively enables fleets of heterogeneous assets to be managed and monitored from a single interface. The interested reader is referred to [7]–[9] and references therein for additional information on underwater networked communications for fleets of AUVs.

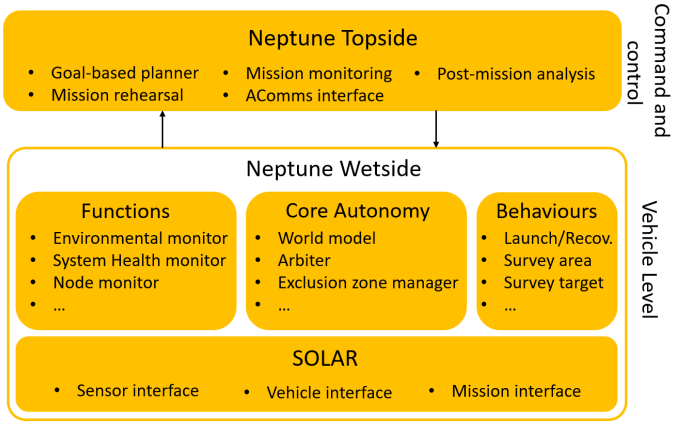

Figure 3. Neptune high-level architecture. Neptune is based on a modular and flexible behaviour-based architecture to separate roles and responsibilities of the various autonomy components.

2.1. Neptune Path Planner

To support the execution of the mission and calculate feasible routes within the area of operations, Neptune implements a search/grid based path planner based on A* [18].

The Neptune planner makes it possible for the vehicle to calculate a safe trajectory that takes into account the vehicle's motion constraints, guaranteeing that collisions are avoided and that the calculated paths are feasible given the specific vehicle dynamics. The planner takes as input: 1) The initial configuration of the robot (where the plan starts from) 2) The goal configuration for the robot to reach (within a threshold) 3) and a map of the environment, which can include excluded areas, obstacles, etc. The final plan is calculated in this work as the path that minimises the distance to reach the goal configuration from the robot initial state.

A plan is composed of as a set of waypoints that can either be sent directly to the vehicle for execution or to MOOS-IvP to be reconciled with additional requirements. While the current plan is executed, Neptune keeps on monitoring the environment and re-plans when needed. Note that in environments where obstacles are continuously moving in and out of the operational area, the planner may frequently update with new optimal paths that are only marginally different from the previous ones, but that will constantly change the trajectory of the vehicle. To avoid sending minor updates, Neptune only sends a new plan when the currently executed one is not feasible anymore (e.g. runs through an obstacle or an excluded area), or the new one is substantially different from the current one (e.g. a moving obstacle has cleared a substantially shorter path). This is done evaluating the Fréchet distance $d_{\mathcal{F}}$ between the two paths (over a discretised number of segments), with the new plan that is sent for execution if $d_{\mathcal{F}}$ is greater than a user specified threshold. Note that while this improves the smoothness of the resulting paths, this also means that trajectories that are not optimal anymore might still be executed. Tuning the acceptable distance between the paths makes it possible to decide the reactivity of the planner to changes in the environment.

3. MOOS-IvP

The surface vessel's autonomy system is based on the MOOS-IvP [11] open source autonomy project. MOOS (Mission Oriented Operating Suite) is a robotic middleware that provides a platform-independent, efficient, and robust architecture for real-time applications components of a robotic system to execute and communicate. MOOS provides a middleware capability based on the publish-subscribe architecture and protocol. MOOS is an open source project maintained by Oxford [12]. The MOOS database (MOOSDB) is central to the architecture [Fig. 4], and provides a mechanism for communication between applications.

MOOS-IvP (Interval Programming) is a MOOS application designed to provide autonomy on robotic platforms and is particularly well-suited to marine vehicles. IvP behaviors determine how the vehicle responds to its environment in pursuit of some defined goal (Fig. 5). Examples include Waypoint behavior (traverse a set of given waypoints) [13] and Avoid Obstacle Behavior (avoid an obstacle based on a tracked object from pObstacleMgr [14]).

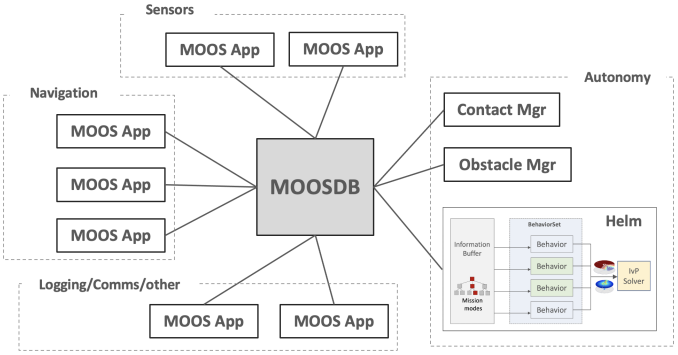

Figure 4. A MOOS community: is a collection of MOOS applications typically running on a single machine each with a separate process ID. Each process communicates through a single MOOS database process (the MOOSDB) in a publish-subscribe manner. Each process may be executing its inner-loop at a frequency independent from one another and set by the user. Processes may be all run on the same computer or distributed across a network. The IvP Helm is a single MOOS app comprised of its own behavior-based architecture. Any app can be replaced with another app that matches the subscriptions/publications interface.

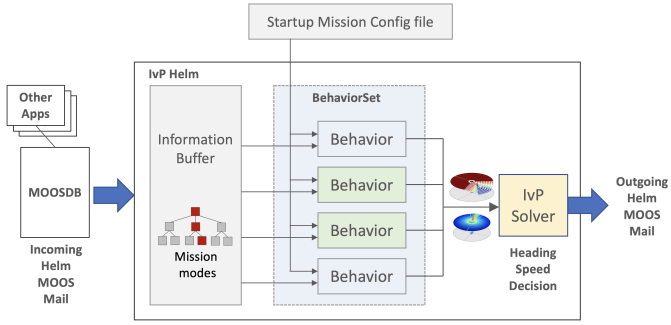

Figure 5. IvP Helm Architecture. 1: Mail messages subscribed for by the helm are read and applied to the local buffer. 2: The helm mode and set of running behaviors are determined. 3: Behaviors execute - posting MOOS variables and an IvP utility function. 4: Competing behaviors are resolved with the IvP multi-objective optimization solver. 5: The helm decision and any behavior postings are published to the MOOSDB.

3.1. Obstacle Avoidance

Obstacle avoidance at its most basic layer is implemented by the MOOS-IvP application pObstacleMgr [14], [15] and the MOOS-IvP behavior AvoidObstacleV21. Objects are detected by the ASV's onboard sensing suite and passed to pObstacleMgr as a convex polygon with an associated ID. The obstacle manager is designed to manage the dynamic incoming clusters to maintain a single list of obstacles. The obstacle manager is configured to produce alerts for the obstacle avoidance behavior based on the ASV range to each obstacle.

Alerts are provided to the helm by pObsticleMgr for creating and modifying each avoid obstacle behavior instance per object. The avoid obstacle behavior will then create a buffer zone around each obstacle as a user-defined convex polygon around the actual polygon obstacle Fig. 6. If the ASVs heading will intersect with the buffer region, then the created objective function will give that heading a negative cost and the vehicle will be guided towards a safer path. The avoid obstacle behavior allows the helm to provide a safe heading around each obstacle based on heading and speed to each obstacle.

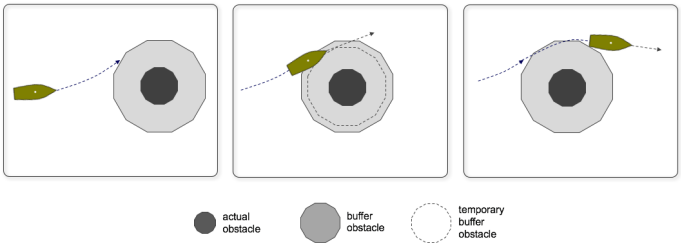

Figure 6. The buffer obstacle may dynamically and temporarily shrink if the robot position intrudes, shrinking until the robot is no longer inside. As the robot opens range, the buffer obstacle will be re-grown back to its original size.

4. Multi-Autonomy Architectures

Having multiple autonomies running on a single vehicle means the system must determine where to delegate authority. Several architectures can be used to support this, and this section briefly summarises the main ways in which multiple-autonomies can be connected together, ranging from a hierarchical autonomy delegation (or master-slave), to having the autonomies running in parallel and relying on a third and external authority (e.g. Arbiter) to reconcile the different decisions.

4.1. Hierarchical Autonomy Delegation

One way to handle the transfer of autonomy is by having autonomies which completely delegate authority to another autonomy system. If only two autonomies are available, this becomes a master-slave architecture. One of the advantages of this method is that once the authority is delegated, there are no conflicts between the two systems, as the vehicle follows commands as issued by the only autonomy which is currently leading. While this method simplifies the interaction between systems, the disadvantage is that there is a limited leverage of the complementary capabilities that the different autonomies may have. More specifically, all the requirements coming from the delegating system need to comply with the capabilities supported by the delegated one. The approach is similar to the known front-seat/back-seat paradigm, and is particularly well suited to configurations where the autonomy engines operate at two different levels, one at a higher, more deliberative level, and one, at lower, more reactive level.

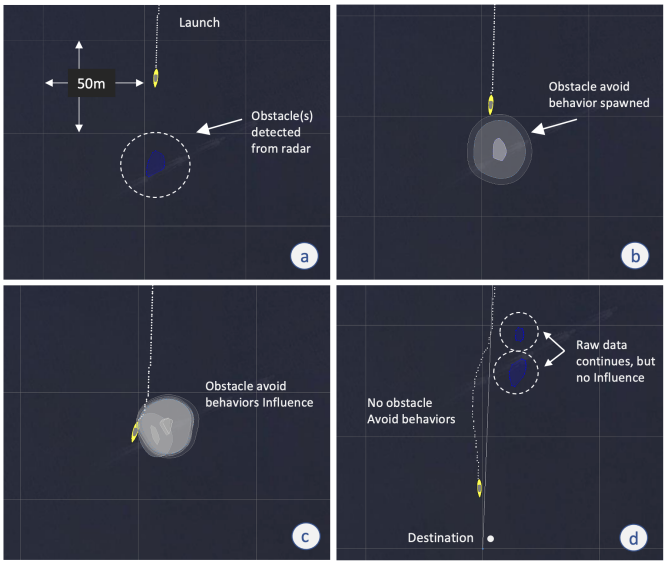

Figure 7. RoboWhaler avoiding an obstacle using the MOOS-IvP behavior AvoidObstacleV21. In (a) the vehicle is approaching the obstacle and radar begins to sense the obstacle. In (b) the vehicle is close enough to the obstacle to spawn an obstacle avoidance behavior. In (c) the vehicle has maneuvered around the obstacle and the behaviors are still actively influencing the vehicle. In (d) the vehicle has dropped the obstacle avoidance behavior due to the relative bearing of the obstacle, but raw data continues to be received for the obstacle. The second distinct obstacle is due to a chase boat near the obstacle.

4.2. Parallel autonomies without external arbiter

A more complex scenario is represented by the case where delegation is not possible due to the lack of required capabilities in one of the two systems. In this case, there is the need for the autonomy engines to effectively run in parallel to control the vehicle and be directly involved in the decision process. The difficulty here is that all behaviours (i.e. the output of every autonomy component) must work at a consistent level of abstraction. Autonomy systems generally differ in the level of abstraction of their control interface, so it becomes necessary to develop a consistent and logical way to decompose the desired level of abstraction (or “plan”) into lower level concepts. In other words, the presence of an interface layer able to glue together autonomies which might provide very different outputs becomes critical. When multiple autonomies are running in parallel at the same time, conflicting requests might be issued to the front-seat computer. In this case, there must exist a formal way to synchronise the requests and deconflict them. One easy way to do that is to rely on an arbiter that can enforce some level of priority.

4.3. Parallel autonomies with external arbiter

One alternative to the solutions presented above can be one that sits somewhere in between the two previous approaches. In this case, some level of arbitration can be used but only to reconcile those behaviors/actions that do not

have a mapping between the two systems, while delegating autonomy when possible. In this case, the two autonomy engines communicate among themselves the information that they can share, and refer to the arbiter only when needed. The advantage is that there is a local disambiguation that can be performed without involving the arbiter, hence reducing the overhead, greatly simplifying the implementation, and the uncertainty of the decision making. The disadvantage is that multiple interfaces are required at different levels, hence limiting the transparency of the final result.

5. Connecting Neptune and MOOS-IvP: Hierarchical delegation with pass-through decision making

This work uses the hierarchical autonomy delegation architecture, where Neptune, with its goal-based architecture, works at a higher level of abstraction than MOOS-IvP to provide a high level plan. Once a plan has been calculated by Neptune, it is then sent to MOOS-IvP to be translated into a sequence of commands for the vehicle to execute. Fig. 8 shows a simplified block diagram of the integration between the two autonomies. The interface block is implemented as two different processes, one living on the Neptune side, and one on the MOOS-IvP side. Each process is responsible for translating requests into service calls, and forwarding the responses to the appropriate autonomy subsystem.

iNeptune is the MOOS driver interface that connects to the equivalent interface realised as a TCP server and hosted by Neptune Wetside (see Fig. 8). These interfaces are responsible for translating autonomy-specific commands to NMEA strings that can then be handled by the corresponding behaviors and applications on the other autonomy. The autonomy tools have been embedded into container images, which allows for easily configurable and replicable deployments. This makes it possible to leverage the existing suite of vehicle drivers to expedite mission development, without needing to spend time developing ports for each potential vehicle. In this way, each autonomy system can work independently while having a clear interface to advertise internal services and access external ones. Additionally, the architecture also supports pass-through decision making. This makes it possible for any architecture to take full control of the vehicle and to directly send commands to the vehicle front-seat for execution.

5.1. Neptune/MOOS-IvP Communications Interface

The communications layer uses TCP sockets, and follows the NMEA 0183 messaging standard. This protocol is well recognized within the marine robotics community, and provided a good compromise between flexibility, simplicity, and readability. We were able to leverage existing NMEA infrastructure and libraries to implement a new suite of messages which formed the core of our communications protocol.

The systems have four messages each that execute very simple operations, but when used together have proven to be incredibly versatile. Neptune's four operations are: Set Mission State (running, stop, passthrough), Go to Waypoints (which includes a list of lat, lon coordinates, as well as a unique ID to identify the sequence), Add Obstacle (which allows MOOS to be aware of obstacles), and Go to Heading at Speed (which allows Neptune to use MOOS as a passthrough, bypassing the IvP Helm's autonomy).

iNeptune provides feedback to Neptune in the form of location and mission telemetry data, and informing of an obstacle's sensed area or removal. This data is provided to iNeptune to in a standard MOOS mailing fashion, allowing for drop-in replacement of new behaviors, different vehicles, or even running Neptune locally without needing to interact with the vehicle (via use of existing MOOS vehicle communications tools).

5.2. Neptune/MOOS-IvP Mission Planning

Mission planning starts by using Neptune Topside and provides mission parameters to MOOS-IvP such as exclusion zones and a waypoint list that represents the mission on a global level. The IvP Helm is configured with default values for mission behaviors and constraints. The default values may be overridden by incoming messages from Neptune, either at the time of mission launch or during mission execution.

5.3. Neptune/MOOS-IvP Path Planning

Path planning is accomplished via a combination of long-term predicting, and short term adjustments. Globally, the mission is planned prior to vehicle launch with Neptune Topside, determining which waypoints the vehicle will visit based on given mission goals and the local environment. Locally, the path of the vehicle is continuously updated by Neptune when there are obstacles or contacts within the current path. Neptune will receive obstacle updates from MOOS-IvP and return an updated path to the current goal waypoint as an updated set of coordinates for the MOOS-IvP waypoint behavior [13] to follow. Lastly, if an obstacle is inadvertently dropped and not reported to Neptune, Neptune deems the variation too small to replan, or a fast moving contact is now within a predetermined distance away from the vehicle before Neptune can replan, then a reactionary MOOS-IvP behavior, AvoidObstacle Fig. 7, will take over and create a new path for the vehicle to follow. Once the obstacle is avoided the vehicle will fall back to following the local path planned by Neptune.

5.4. Pass-through Decision Making

Creation of a pass-through decision making architecture was accomplished by adapting the Neptune-MOOS-IvP bridge to allow Neptune to control the vehicle directly. All MOOS-IvP decision making behaviors are disabled in this mode. This mode was developed to show that Neptune and

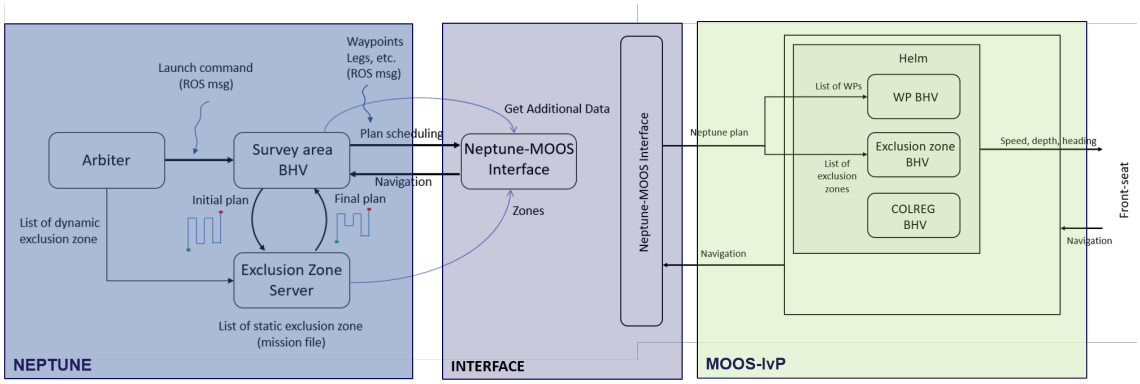

Figure 8. Neptune-MOOS-IvP integration using hierarchical authority delegation. Although, not shown in the picture, the architecture also supports pass-through decision making whereby Neptune can control the vehicle directly sending command to the front-seat.

MOOS-IvP can work independently from one another as needed, therefore making Neptune and MOOS-IvP modular and complementary to vehicle control in the end. In the pass-through mode MOOS-IvP is still connected to the vessel, but is acting to facilitate passing desired heading and desired speed commands from Neptune to the vehicle under control as well as passing vehicle perception and navigation information to Neptune.

While in pass-through mode, Neptune is responsible for ingesting vehicle navigation information as well as obstacles that are sent by the perception software. Based on all pertinent information, Neptune creates a safe path around any obstacles to the goal waypoint, and sends a heading and speed command to the vehicle, which is forwarded to the front-seat controller.

6. Results

6.1. Neptune/MOOS-IvP Simulation

Extensive simulation of the Neptune-MOOS-IvP bridge was accomplished to test for compatibility and bugs within the system. Simulation of how each system performs as they are deployed helps eliminate many hours of on-water testing that can be time consuming and expensive. Simulation of Neptune planning around an obstacle field with MOOS-IvP in control of an ASV is outputted as expected in this mission. The bridge between MOOS-IvP and Neptune is the iNeptune application shown in the blue appcasting output of pMarineViewer Fig. 9. A zoomed in view of the iNeptune output Fig. 10 shows the messages being sent and received. The report also includes the reporting events depicting visited and updated waypoints as well as obstacles detected.

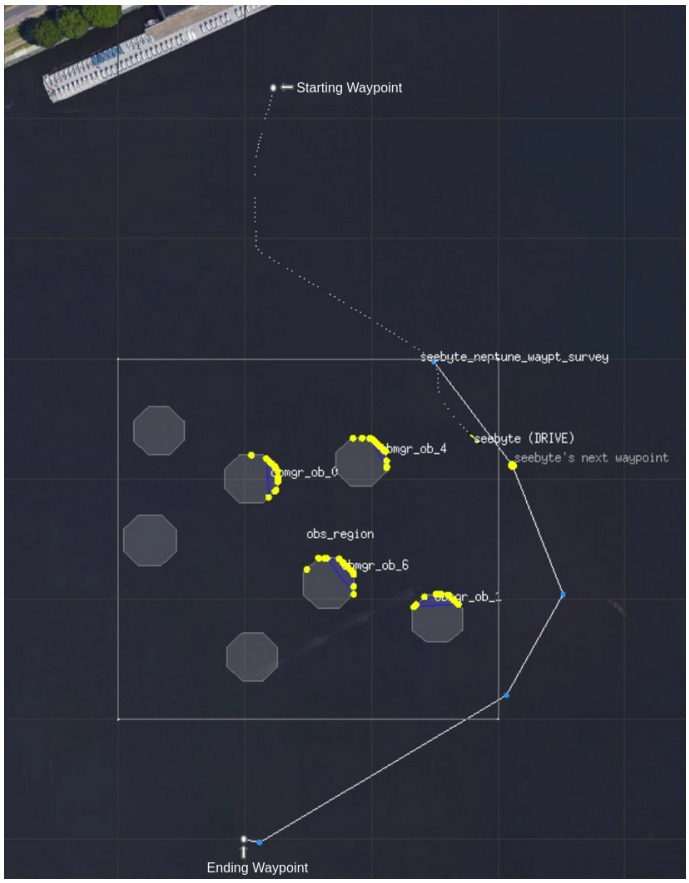

Figure 9. Simulation of Neptune and MOOS-IvP displayed on pMarineViewer. The simulated obstacles are detected by the vehicle (yellow dots around obstacles) and reported to Neptune for an updated path to the ending waypoint.

6.2. Neptune/MOOS-IvP Experimental Results

Experimental testing was accomplished on board the RoboWhaler. RoboWhaler is an MIT AUV Laboratory and Mercury Marine developed autonomous surface vessel. For a specific explanation of the vessel's hardware and software

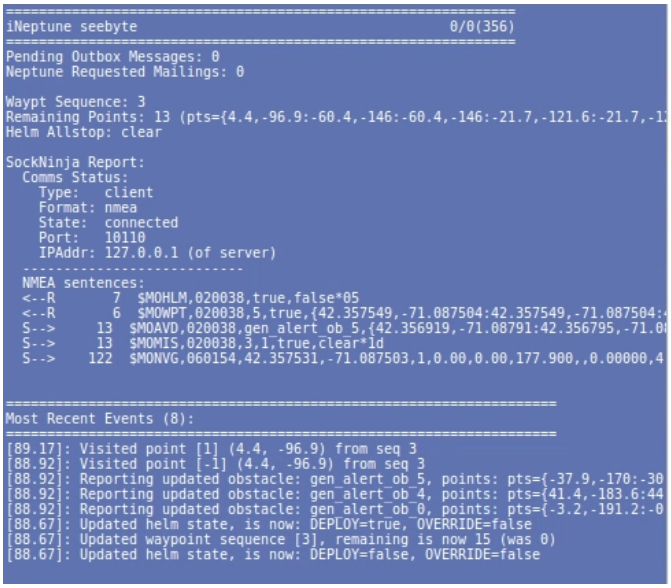

Figure 10. Zoomed in appcasting report from iNeptune in pMarineViewer. MOOS is able to display the connection to Neptune, the last five NMEA messages passed, as well as any of the most recent events between Neptune and MOOS-IvP

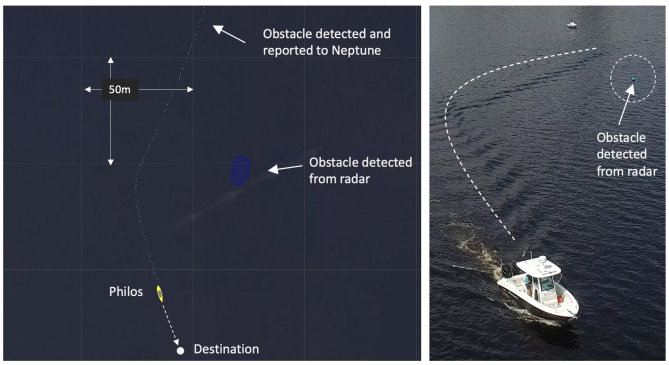

Figure 11. Captured data playback in MOOS-IvP of Neptune execution of pass-through mode. Neptune is receiving obstacle reports from MOOS-IvP, but no MOOS-IvP behaviors are invoked when performing the mission.

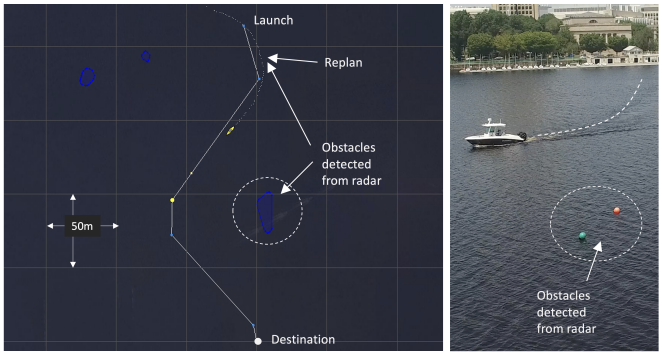

Figure 12. Captured data playback in MOOS-IvP of Neptune shared mission path planning. MOOS-IvP reporting obstacles to Neptune and Neptune is returning a new path to avoid the obstacles and proceed to the goal waypoint. Obstacles are represented by the blue polygons. After Neptune provides the new updated path, MOOS-IvP resumes control of processing the path waypoints.

suite, see "RoboWhaler: A Robotic Vessel for Marine Autonomy and Dataset Collection" [16] in the OCEANS 2021 conference.

The basic setup of the missions are to test how well the systems handle path planning around both static obstacles and slow moving contacts. The global mission plan is to autonomously transit from one side of the river to the other via two waypoints on each side of the river. Static obstacles were provided by a few strategically placed buoys in the ASVs path. Slow moving contacts were provided by an engineer driving a skiff towards the path of the ASV during transit, and a few opportunistic kayaks that happened to be passing through our test area.

6.2.1. Pass-Through Decision Making. As previously mentioned in 5.4 the primary objective of the pass-through decision making mode is to show that Neptune can directly control RoboWhaler without any influence from another autonomy system when desired. In this case no MOOS-IvP behaviors, which are designed to control the vehicle, are invoked while in pass-through mode. Neptune is able to send heading and speed commands to the primary vehicle computer with MOOS-IvP acting as an intermediary to the vehicles control computer. Fig. 11 is an image captured from the recorded output of MOOS-IvP during a pass-through mission on RoboWhaler. The original path that the ASV is pre-programmed to take is through the obstacle represented by the blue polygon. While in transit, Neptune is receiving obstacle and vehicle navigation reports from MOOS-IvP and sending back a desired heading and desired speed to command RoboWhaler safely around the obstacle in its path. The new path is seen by the white trail that is left as a visual artifact while Neptune safely navigates RoboWhaler around the obstacle.

6.2.2. Path Planning. On-water tests were performed to test how complementary autonomy software could interact and safely plan a path as obstacles come into view of the vehicle. As mentioned in Section 5.3, during the mission Neptune will continuously update the vehicle path based on whether obstacles or contacts are within the current path. MOOS-IvP is also in the loop to monitor and take control of the vehicle in emergency situations. Fig. 12 is an image captured from the output of MOOS-IvP during a mission to test Neptune and MOOS-IvP interoperability. The original path would have brought the vehicle directly through obstacle field of buoys seen in Fig 12. Once RoboWhaler detects the obstacles, they are reported to Neptune and Neptune sends and updated waypoint list to MOOS-IvP to follow. The updated path is shown by the solid white lines with breadcrumb waypoints for RoboWhaler to follow. Neptune safely plans a new path around the obstacle field as well as a pair of kayaks on the river.

7. Conclusions, Ongoing Efforts

This paper described how it is possible to integrate together multiple autonomy systems onto a single vehicle, switching between different autonomy architectures based on mission requirements. SeeByte's Neptune and MIT's MOOS-IvP have been integrated using a hierarchical delegation of authority between the two systems, together with a pass-through decision making that makes it possible for each autonomy to retain exclusive authority at any point during the mission. Results of this integration have been presented via simulations and in-water tests where the systems was deployed on MIT's RoboWhaler and demonstrated in a safe crossing scenario. Future work will focus on how to further generalise the implemented architecture to support any number of architectures and how to implement a generic service oriented architecture that can be used to tackle even more complex scenarios. This might include sharing and synchronizing world-models across multiple architectures and allowing for real-time multi-objective optimisation. This can be based on known and predefined conditions available before the mission starts, and/or based on ad-hoc services and capabilities which are required as the mission evolves. Finally, an interesting problem would be how to scale the approach from single vehicles to multi-vehicles scenarios where behaviours or capabilities can reside on any of the available assets.

8. Acknowledgments

SeeByte would like to thank Jean-Baptiste Doyon for his work to set-up the initial Neptune to MOOS bridge and Jonatan Willners for his help to adapt the Neptune Path Planner to the safe crossing scenario.

MIT would like to thank Brunswick Corporation and our partnership with the Emerging Technologies and Innovation Processes group at Mercury Marine (a division of Brunswick Corporation). The loan of the Boston Whaler (R/V Philos), prior funding, and engineering expertise made it possible to outfit the vessel and interface into the vehicle control system, which was the beginnings of RoboWhaler.

This work has been partially funded by the Office of Naval Research (ONR) under contract n. N00014-19-C-2056. The MOOS-IvP autonomy research and open source project has been supported by the Office of Naval Research (ONR) Code 311 under award n. N00014-19-1-2180.

References

[1] Ferri*, G., Munafò*, A., Tesei, A., Braca, P., Meyer, F., Pelekanakis, K., Petroccia, R., Alves, J., Strode, C. and LePage, K. (2017), Cooperative robotic networks for underwater surveillance: an overview. IET Radar Sonar Navig., 11: 1740-1761. https://doi.org/10.1049/iet-rsn.2017.0074

[2] Munafò, A., Simetti, E., Turetta, A. et al. Autonomous underwater vehicle teams for adaptive ocean sampling: a data-driven approach. Ocean Dynamics 61, 1981–1994 (2011). https://doi.org/10.1007/s10236-011-0464-x

[3] G. Ferri, A. Munafò and K. D. LePage, "An Autonomous Underwater Vehicle Data-Driven Control Strategy for Target Tracking," in IEEE Journal of Oceanic Engineering, vol. 43, no. 2, pp. 323-343, April 2018, doi: 10.1109/JOE.2018.2797558.

[4] Caiti, A., Calabrò, V., Munafò, A., Dini, G. and Lo Duca, A. (2013), Mobile Underwater Sensor Networks for Protection and Security: Field Experience at the UAN11 Experiment. J. Field Robotics, 30: 237-253. https://doi.org/10.1002/rob.21447

[5] C. I. Sprague, Ö. Özkahraman, A. Munafo, R. Marlow, A. Phillips and P. Ögren, "Improving the Modularity of AUV Control Systems using Behaviour Trees," 2018 IEEE/OES Autonomous Underwater Vehicle Workshop (AUV), 2018, pp. 1-6, doi: 10.1109/AUV.2018.8729810.

[6] Y. R. Petillot, G. Antonelli, G. Casalino and F. Ferreira, "Underwater Robots: From Remotely Operated Vehicles to Intervention-Autonomous Underwater Vehicles," in IEEE Robotics & Automation Magazine, vol. 26, no. 2, pp. 94-101, June 2019, doi: 10.1109/MRA.2019.2908063.

[7] E. M. Sozer, M. Stojanovic and J. G. Proakis, "Underwater acoustic networks," in IEEE Journal of Oceanic Engineering, vol. 25, no. 1, pp. 72-83, Jan. 2000, doi: 10.1109/48.820738.

[8] J. Sliwka and R. Petroccia and A. Munafò and V. Djapic, "Experimental evaluation of Net-LBL: An acoustic network-based navigation system," OCEANS 2017 - Aberdeen, 2017, pp. 1-9, doi: 10.1109/OCEANSE.2017.8084794.

[9] A. Caiti et al., "Linking Acoustic Communications and Network Performance: Integration and Experimentation of an Underwater Acoustic Network," in IEEE Journal of Oceanic Engineering, vol. 38, no. 4, pp. 758-771, Oct. 2013, doi: 10.1109/JOE.2013.2279472.

[10] Hastie, H and Liu, X and Patron, P and Petillot, Y 2017, 'Talking Autonomous Vehicles: Automatic AUV Mission Analysis in Natural Language', Paper presented at 60th MTS/IEEE OCEANS Conference, Aberdeen, United Kingdom, 19/06/17 - 22/06/17

[11] Michael R. Benjamin, Henrik Schmidt, and John J. Leonard. The moos-ivp open source marine autonomy project. http://www.moos-ivp.org (2017).

[12] Paul Newman. The moos open source robot middleware project. http://www.themoos.org.

[13] Waypoint Behavior: https://oceanai.mit.edu/ivpman/bhvs/Waypoint.

[14] pObstacleMgr: https://oceanai.mit.edu/ivpman/apps/pObstacleMgr.

[15] Michael R Benjamin, Michael DeFilippo, Paul Robinette, and Michael Novitzky. "Obstacle avoidance using multi-objective optimization and a dynamic obstacle manager". In: IEEE Journal of Oceanic Engineering 44.2 (2019), pp.331–342.

[16] Michael DeFilippo, Michael Sacarny, and Paul Robinette. "RoboWhaler: A Robotic Vessel for Marine Autonomy and Dataset Collection". In: OCEANS 2021 San Diego. IEEE. 2021. In Press.

[17] Paul Robinette, Michael Sacarny, Michael DeFilippo, Michael Novitzky, and Michael Benjamin. "Sensor Evaluation for Autonomous Surface Vehicles in Inland Waterways". In: OCEANS 2019 Marseille. IEEE. 2019, pp. 1–8.

[18] Hart, P. E. and Nilsson, N. J. and Raphael B., "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," in IEEE Transactions on Systems Science and Cybernetics, vol. 4, no. 2, pp. 100-107, July 1968, doi: 10.1109/TSSC.1968.300136.