The LSTS Toolchain for Networked Vehicle Systems*

José Pinto[1], Paulo S. Dias[1], Ricardo Martins[1], João Fortuna[1], Eduardo Marques[2] and João Sousa[1]

***Abstract*— This paper describes the open-source software toolchain developed by the Underwater Systems and Technology Laboratory (LSTS) for supporting networked heterogeneous air and ocean vehicle systems. The toolchain supports the deployment of air and ocean vehicles interacting over limited acoustic and wireless networks combined with disruption-tolerant networking protocols. We present the different components of the toolchain and how they can be deployed and extended for different scenarios. We conclude with descriptions of recent applications to onboard deliberative planning and integration of low-cost micro UAVs into the toolchain.**

I. Introduction

There is an ongoing trend towards the creation of Networked Vehicle Systems. These systems are composed by autonomous vehicles, sensors and human operators that form a network with a dynamic topology. This happens because each node in the network has a set of communication means which bandwidth, latency and reliability depends on their pose and the poses of its peers. Each node in the network may have a dynamical physical position. As nodes move, the network topology changes: communication will be affected in terms of connectivity, bandwidth and reliability. We are thus interested in complex behaviors that require orchestration of communications, movement and computations inside the network.

Consider the scenario in Fig. 1. Two operators are connected to the network (others could join in at any moment) and control the network through communication links that are created and destroyed dynamically. If the objective of the operator on the right was to obtain an underwater sidescan survey at a remote location, this should be stated to the system and the network should automatically adapt to encompass the resulting objectives. The objectives would include:

- find one or more vehicles which are capable of gathering the required data;
- survey location using one or more vehicles; and
- relay (important pieces of) data back to base station.

In the presented scenario one UAV could, for instance, download the objectives from the control station, fly towards a device that provides both wireless and acoustic underwater communications and relay the objectives to that device. This device is then responsible for contacting one or more AUVs that are capable of doing the survey and relaying the objectives to them. After the survey is completed, the data needs to somehow travel back to the base station using direct communications (if the AUV comes near the base station) or using one or more communication relays like before.

Orchestrating mobile heterogeneous nodes like this is very complex for human operators since they must be aware of all connected and disconnected devices, their current capabilities and state and also any faults or changes that may occur in the system. This is why it is very important that the system aids the operators in decision making and allow disconnected devices to sense the environment and adapt behavior to fulfill the desired objectives.

At LSTS, we have developed different algorithms and technologies for the operation of these networked vehicle systems, with applications to adaptive ocean sampling [1], mine hunting [2], data muling [3], among others [4]. For this we have developed several AUVs, UAVs, ROVs and ASVs (see figure 2). Most of these vehicles have been developed from scratch by the group including their electronics, onboard software and operating consoles. Others have been developed under ongoing collaborations with the Portuguese Air Force Academy (bottom-left UAV in figure 2) and the Portuguese Navy (bottom-right AUV in figure 2).

This paper presents the open-source LSTS software toolchain [5] and design decisions that were taken to enforce the modularity and adaptability of these tools. In section II we describe our system architecture and illustrate it with a sample deployment scenario. We then present the onboard software in III and the integrated Command and Control Software in more detail in section IV. In sections V and VI we describe how this toolchain was extended for supporting

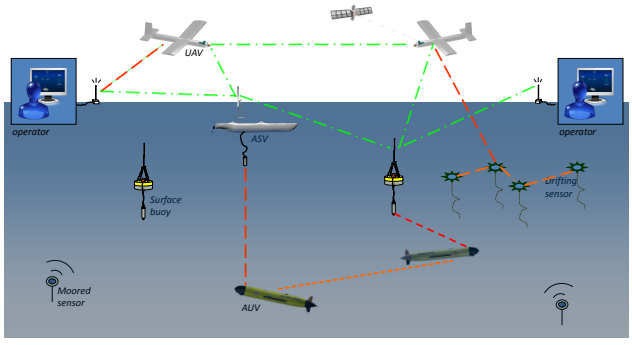

Fig. 1. Conceptual deployment of a Networked Vehicle System

[1] Underwater Systems and Technology Laboratory (LSTS), Faculdade de Engenharia da Universidade do Porto lsts@fe.up.pt

[2] Large-Scale Informatics Systems Laboratory, Faculdade de Ciências da Universidade de Lisboa

**The research leading to these results has received funding from the European Commission FP7-ICT Cognitive Systems, Interaction, and Robotics under the contract #270180 (NOPTILUS).*

**The authors gratefully acknowledge Kanna Rajan and Frédéric Py from MBARI for discussions on autonomy and support in integration of TREX.*

Fig. 2. Vehicles supported by the LSTS toolchain

Micro Aerial Vehicles and onboard deliberative planning, respectively, and report results from field deployments in VII. We end with conclusions and lines of future work in VIII.

II. Architecture

In order to be possible for a set of users to control a fleet of heterogeneous vehicles and sensors the underlying architecture must be flexible enough that it can encompass not only very diverse vehicle hardware but also communication means, mission scenarios, operator expertise among others.

Our approach is to allow access to the different systems through explicit and progressively lower level control / supervision layers. This way it is possible to have system interoperability at different levels and have controllers for low-level actuations as well as multi-vehicle operations (and all in between). This can be seen in figure 3, where we see that maneuver-level and plan-level control can both be done onboard the vehicle (fully autonomous operation) or it can be controlled by an external operator console while the vehicle is within communication reach.

A. Communications

All the components in our system share the same communication protocol: the Inter-Module Communications protocol, described in [6] and available from [7]. IMC is a message-oriented protocol used for both inter-process communication, inter-vehicle and operator-vehicle communications, logging and also dissemination to the web. The entire

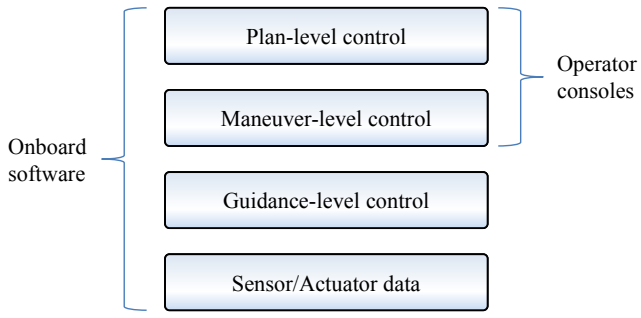

Fig. 3. Different control layers and systems they reside in

IMC protocol definition and its documentation is documented in a single XML from where language bindings are generated. This flexibility in creating new types of messages/events and listening to them onboard and/or offboard is one of the cornerstones of our system. We are currently using IMC on unmanned vehicles, data loggers, communication gateways, portable devices, laptops and web servers.

Moreover, since the protocol is agnostic from the underlying communication mean, it can also be used in disruption-tolerant networks by employing a DTN implementation.

In our toolchain, all processes and devices communicate by exchanging IMC messages. These messages can represent either chunks of a data stream (messages originated in sensors or component states) or they represent an asynchronous event that should be handled by other component.

All IMC messages contain a header with information like its type, version, timestamp, origin and destination. Origin and destinations are encoded as a two-part identifier: the physical component and the computational component. A message can thus be addressed to a set of devices of a certain group (all consoles, all AUVs, ...) or they can be addressed specifically to a certain process running onboard a specified device.

Communication gateways are able to forward messages from one communication mean to another. For instance, the Manta communications gateways are able to translate messages received by acoustic modem into messages that are transmitted encapsulated inside UDP datagrams through Wi-Fi. These communications gateways are used in the operation of AUVs where they are connected to one or several operator consoles and allow near real-time communication with untethered underwater vehicles while they operate and, at the same time allow long-range communication with ASVs, AUVs at surface or nearby UAVs.

In figure 4, the communications diagram of a typical operation scenario is depicted. An UAV (1) may be connected to the base station through a communications gateway (2) while operating. Moreover, in the base station one or more operators are connected to the system via laptops (4) or portable handheld devices (5) which are useful when the operators require mobility like during tele-operation.

The operators on the shore can command the UAV or the AUV (3) which are both connected to the gateway but, in the case of the AUV, it can use only acoustic communications while underwater and when at the surface it can use both Wi-Fi or the acoustic modem. The communications gateway also contains a 3G GSM modem that, when active can be used to forward messages to a remote web server (8) that stores incoming data. This data can be retrieved and visualized in real time by any other system that is connected to the web (7).

In terms of control, while the vehicles are connected to the base station, the operators can issue both maneuver commands or plan commands. While the vehicles are disconnected they should autonomously decide which commands should be done next, according to their current state and objectives.

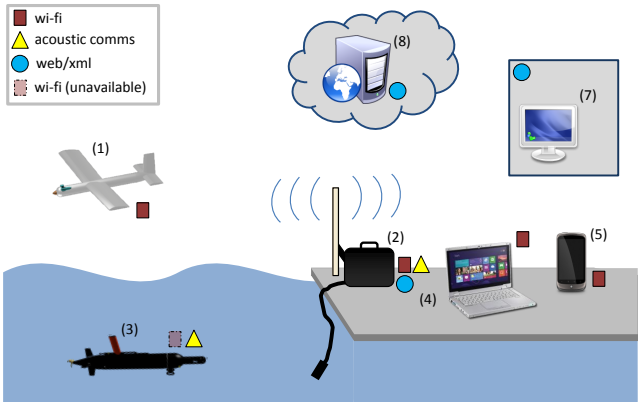

Fig. 4. Components deployed in a typical operation

III. Onboard Software

The onboard software used in all our embedded systems is DUNE (DUNE Uniform Navigation Environment) [8]. DUNE is writen in C++ and runs on Linux 2.6+, QNX v6.x, Solaris, Mac OS X, eCos, RTEMS, OpenBSD, FreeBSD, NetBSD and Microsoft Windows.

A. DUNE Tasks

In DUNE, related logical operations are isolated from each other in tasks which usually run in separate threads of execution. Tasks communicate with one another only by using a message bus which is responsible for forwarding IMC messages from the producer to all their registered receivers. Each task follows a common life-cycle and also has method handlers for all messages that it consumes. The life-cycle of a DUNE task comprises the following methods:

- onEntityReservation() / onEntityResolution()
- onReportEntityState()
- onResourceAcquisition() / onResourceRelease() / onResourceInitialization()
- onUpdateParameters()
- onRequestActivation() / onRequestDeactivation()
- onActivation() / onDeactivation()
- onMain()
- consume($< M >$ message)

Any implemented task can choose to override the inherited (empty) implementation of the previous methods with their specific code. As an example, a sensor payload implements the methods *onActivation* and *onDeactivation* so that it turns on and off the payload as requested by other tasks and thus save on energy consumption. Resource aquisition and initialization also follow timed phases in the initialization of the system.

Each task can also register one or more computational entities that can later be used to address it univocally inside a networked vehicle system. Tasks can also implement *onUpdateParameters* that is triggered whenever their configuration gets changed.

The *onMain* method is called periodically by the scheduler so it is overriden by any tasks that want to execute something at timed delays. Finally, all tasks can consume messages generated by any other threads by implementing *consume* methods that take as their sole parameter the generated message (handled messages will have a corresponding type).

In DUNE, tasks are divided into sensors, actuators, estimators, monitors, supervisors, controllers and transports as follows:

- *Sensors* are device driver tasks, associated with some hardware that measures the environment.
- *Actuators* are device drivers for hardware that allows the vehicle to interact with the environment and / or move.
- *Estimators* are tasks that aggregate information from sensors into state estimations. One good example of an estimator is the *Navigation* task.
- *Controllers* are tasks that handle high-level commands and transform them into lower-level commands or actuations according to current estimated state. For instance, all supported maneuvers have a corresponding maneuver controller.
- *Monitors* are tasks that receive information from other tasks and may change the vehicle state accordingly. For instance, the *Operational Limits* monitor will change the vehicle mode to “blocked” whenever the operational limits are breached.
- *Supervisors* are tasks that enable and disable other tasks according to the current vehicle state. For instance, if the vehicle enters “blocked” mode, the vehicle supervisor will stop the current maneuver controller from sending commands.
- *Transports* are tasks in charge of transporting messages in and out of the message bus. Logging is a special transport task that listens to a set of messages and records their serialized state to persistent storage. Other transport tasks are *UDP*, *TCP*, *HTTP*, etc.

B. Run configurations

All running instances of DUNE share the same code base but run under different configurations. We call configuration to a description of the tasks that are enabled and their parameters. The configuration of a running DUNE instance can change according to user intervention or in response to a *Supervisor* task command.

A DUNE configuration file describes which tasks are enabled initially and also what are their initial parameters. Using a referencing mechanism, a configuration file may include parts of other configuration files. This allows the creation of small and less error-prone vehicle-specific configuration files, since most of the tasks (and their parameters) are common between similar vehicles.

To choose if a task will be enabled in a configuration file, the user must select in which profiles the task will be enabled by default. DUNE uses profiles to allow multiple (typical) configurations to be defined on a single file. Some examples profiles are:

- *Hardware* This task will be enabled only when DUNE is connected to the real hardware devices (real vehicle).

- *Simulation* These tasks are enabled only when DUNE is running with no connection to real hardware sensors and actuators. Simulated versions of the sensors and actuators will thus correspond to simulation tasks that run only in this profile and produce simulated data.
- *HIL* This profile is used on the real hardware but part of the actuators and sensors will have simulated inputs / outputs. For instance, the *Thruster* task in this mode will run at a fraction of the commanded RPMs so that it can be used safely out of the water.

In the configuration file, tasks are enabled/disabled by selecting between *Hardware*, *Simulation*, *HIL*, *Always* and *Never* profiles of execution.

C. Safety mechanisms

DUNE features a set of tasks (Monitors) tasks that constantly check vital parts of the system. In case any of these tasks enter an error state, a listening Supervisor will enable / disable other tasks accordingly. We next give examples of these safeties for underwater and aerial vehicles.

1) Fuel Level: If the *Fuel Level* task detects that the fuel is too low, the vehicle supervisor will detect it and stop the current plan controller from executing, replacing it by a safety maneuver controller.

2) Leaks: If there is a leak detection onboard an AUV, its vehicle supervisor will shutdown all payloads and execute a safety maneuver to bring the vehicle to the surface.

3) Operational Limits: If the user-defined limits were breached by the current plan controller, it will be blocked from controlling and only teleoperation and safety maneuvers are allowed to be executed by the vehicle supervisor.

4) Communications: If an UAV loses communication with the base station for more than a user defined time interval, the vehicle will stop the current plan and execute a predefined plan that will bring it back near to the base (hover next to the base station).

IV. Command and Control Software

Neptus [9] is the C2 (Command and Control) software used to command and monitor our unmanned systems. It is written in Java and it currently run in Linux and Microsoft Windows operating systems.

The main Neptus communication interface is IMC, making it interoperable with any other IMC-based peer. Neptus has been used to command all our systems which correspond to very heterogeneous classes of autonomous vehicles and sensors.

Despite the heterogeneity of the controlled systems, Neptus provides a coherent visual interface to command all these assets. The main purpose is for an operator to take advantage of what these assets have to offer in terms of sensor and actuator capabilities without having to dwell into specific C2 software and details.

A typical mission life-cycle comprises the planning, execution, and review and analysis phases:

1) The *planning phase* is generally performed prior to the execution of a mission. On it, the operator is generally equipped with the mission objectives and its where he becomes acquainted with possible obstacles, depths, tides, traffic, etc. With these elements in mind the operator can choose the best locations for the command center, communication and location aids, and starts preparing the mission plans to be used and does rough simulations of them.
2) In the *execution phase*, the operator is in charge of preparing the vehicles for deployment, monitor the systems telemetry and execute/adapt the mission plans. Also, in a multi-vehicle operation, the several C2s need to be aware of each other and cooperate to achieve the global mission objectives.
3) The *review and analysis phase* takes place on site or after the mission is concluded. In this phase, the collected data is processed and analyzed to compile the mission results or evaluate individual plan execution in order to adjust and re-plan to achieve another desired outcome.

Neptus is a framework that was created from scratch having in mind its adaptability and flexibility to encompass needs from diverse vehicles, scenarios and operator experiences. As a result, it provides the rapid creation of derived tools and can be customized according to operator and mission needs.

A plug-in can be developed independently of the main Neptus source and added as a compiled jar file. This way Neptus can be extended by a third party with new components with the added possibility of not sharing source code among developers (which can sometimes be a requirement for ITAR-constrained code, for instance).

A. Neptus Operator Consoles

An operator console is a Neptus application that provides basic functionality to its plug-ins: a communications infrastructure, several layout mechanisms, means for showing notifications to the user and a map that can be extended with new layers and interaction mechanisms, among others. The resulting consoles support all the phases of the mission life-cycle in an integrated interface (mission revision is intentionally simplified in this interface).

A console is defined by an XML configuration file that lists the plug-ins and their configuration (desired layout in the window, specific parameters), following the pattern used by DUNE for describing running configuration instances.

There are several options to configure the placement of the components, but most important it is the ability for every console configuration to hold several layouts of its components. These layouts are called profiles, again resembling the profiles from DUNE but this time they are usually switched at run time by the user.

Each console profile can show and hide selected components declared in the console and define a distinct placement for them. These profiles allow a quick change of components layout allowing the operator to choose the one that is better adapted to the current mission phase, a currently focused system or abnormal situation.

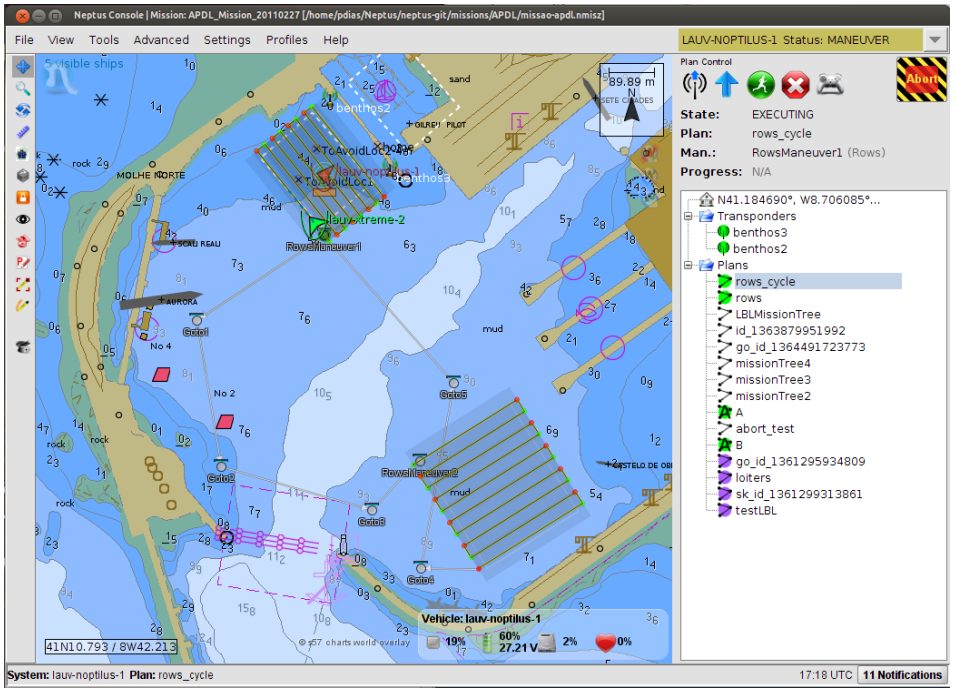

Fig. 5. Neptus operator console

The console shown in figure 5, is a console targeted at operating the LAUV AUV. We can look at this console by dividing it in two, the left and right sides.

On the left we see the map. In this component we can find the virtual representation of mission site. For the map tiles, the user can choose from several sources like Open Street Map, Google Maps, or Virtual Earth (tiles are cached for offline use). *S57* nautical charts are also available but for this option, nautical charts are loaded from disk while on the former they are loaded from the Web.

On the right-hand side of the interface, the console provides several planning components and commands. On the top, a combo box makes it possible to change the system currently being controlled. Below, the plan control buttons allow operators to send and execute mission plans and monitor the state of execution of active plans in the panel below. The tree in right is called the mission tree and there is where the operators can find and edition the mission elements (beacon locations, mission plans and static base location). These elements allow a suitable command of the AUV.

Moreover, several less used widgets can be shown/hidden in pop-up dialog windows by using accelerator keys. This was added in order to access more detailed information quickly as needed.

In the map panel in the figure 5, we can see a mission plan which consists of two rows (lawnmower pattern) interleaved with go-to maneuvers. When editing a plan, the operator can select from the set of the vehicles' feasible maneuvers and build a plan by adding maneuvers and creating transitions between them (Neptus creates a sequencing transition as default). For each maneuver, a set of maneuver-specific parameters, together with vehicle-specific payload parameters can be set in order to use, or disable, vehicle payloads or configure some other parameter of the vehicle.

Our UAVs are also controlled by Neptus through a similar interface.

B. Communications infrastructure

The communications infrastructure in Neptus supports messages coming from *UDP* (unicast and multicast) or *TCP* as well as connecting to web services (*XML/HTTP*). In order to establish a connection to a system, a common communication mean must be found and set. Neptus holds some vehicle configurations that are updated by a discovery mechanism based on the multicast of *Announce* IMC messages.

The *Announce* message describes the originating system by providing its system name and type, current location and set of available services (e.g. communication protocols or gateways, or onboard payloads). These services are defined by a URI. Examples of these URI are: *imc+udp://192.168.106.34:6002/* (IMC via UDP), *imc+tcp://192.168.106.34:6002/* (IMC via TCP), or *dune://0.0.0.0/uid/1294925553839635/* (DUNE instance ID, changed on DUNE reboot).

Neptus communications infrastructure receives this information (while also publishing Neptus own services) from all reachable systems. After a system is known to Neptus, the operator can start exchanging telemetry and control data with that system, on request.

In the case of UAVs we also made a partial implementation [10] of the STANAG 4586 protocol [11]. NATO Standardization Agreement 4586 is a Standard Interface of the Unmanned Control System (UCS) for NATO UAV Interoperability.

To add support for 4586, Neptus was extended with new specific console widgets and the communications infrastructure was adapted to be able to exchange 4586 messages. This was done by extending the base communications infrastructure.

C. Mission Review and Analysis

The Neptus interface for analyzing the collected data is the MRA (Mission Review and Analysis) shown in figures 6 and 7. This interface is targeted at viewing IMC logs and other collected sensor data (for sensors that have own specific file formats).

Logs are produced onboard devices by serializing generated messages and concatenating them into one or more binary files. In order to present the data, these files are initially indexed and merged with files logged onboard other systems, ordering them by the origin timestamp. As a result, both operator commands and multi-vehicle surveys can be concentrated in a single log file and visualized (see section VII-A for multi-vehicle visualization examples).

MRA interface is divided in two parts. On the left side, the messages and available visualizations are listed, while on the right the active visualization is displayed. Any IMC message can be inspected rapidly in a tabular form or as a multi-variable/multi-message time-series plot. Some plots are predefined and others can be created inside MRA by selecting fields to plot. Moreover, plots can also be created in Javascript derived language and saved as a plot script.

Other specialized visualizations are also available, and similarly to the console widgets, can be added as Neptus plug-ins. Examples of these specialized visualizations are the sidescan analyzer seen in figures 6, 12, or 14, plots like the figure 10, 13, or 15, mission replays with map overlays like seen in figure 11, a photo visualization plugin like in figure 7, 3D bathymetry and trajectory visualizations, etc.

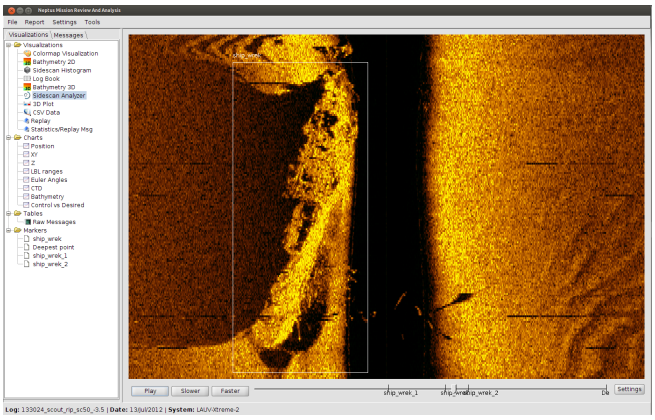

Fig. 6. Sidescan visualization in Neptus MRA

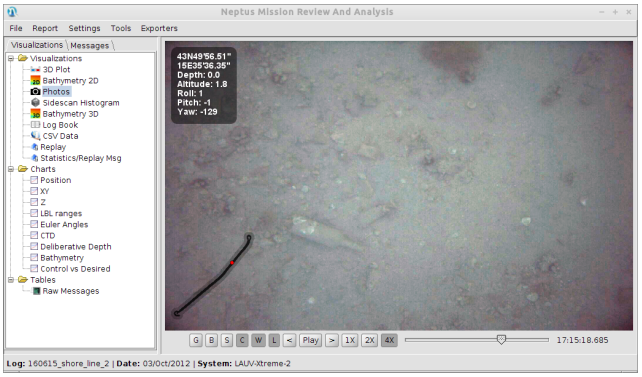

Fig. 7. Camera visualization in Neptus MRA

V. Extensions for Micro Aerial Vehicles

The prices of UAVs recently have come down abruptly mostly due to the creation of a very low-cost auto-pilot hardware and software based on the Arduino platform, the ArduPilot [12].

In order to add support for vehicles based on Arduino, some extensions were added to DUNE in order to be able to parse normal plans and transform them into ArduPilot commands. For navigation and low-level control of Micro Aerial Vehicles we use ArduPilot-Mega, while for plan and maneuver execution we rely on DUNE.

If we compare this hybrid ArduPilot-DUNE system to just ArduPilot, in the former we get rich features such as advanced behavior and missions. Comparing it to a single system with low and higher level control this also presents advantages like fault-tolerance to an error on the main computer.

A. Hardware integration

ArduPilot is integrated with no need for modifications in terms of its firmware or hardware as this board is connected without modifications to DUNE which is running on a separate CPU stack (IGEPv2 [13]). DUNE provides communications through Wi-Fi with the base station and controls the UAV by sending waypoints to be followed by the auto-pilot. This hardware configuration was used for both MAVs in figure 8.

B. Control integration

In order to integrate waypoint control of the ArduPilot, a new DUNE task was created that translates guidance commands into ArduPilot waypoints. Moreover, this task

Fig. 8. ArduPilot-based MAVs integrated with the LSTS toolchain

also translates between ArduPilot telemetry data and corresponding IMC messages.

Since the integration was done at the guidance level, all plans / maneuvers that use guidance commands to control the underlying hardware can still be used.

VI. Extensions for Onboard Deliberative Planning

In order to provide our toolchain with deliberative planning capabilities, we integrated already existing tools: the TREX [14] teleo-reactor executive and the EUROPA [15] planner. As seen in [14] TREX interfaces the EUROPA planner by a set of shared timelines to where different reactors (components) post observations and goals.

On the existing toolchain, a new planning interface was created as a new Neptus console plug-in, a new a TREX monitor task was added to DUNE and two new messages were added to the IMC protocol. Results of this integration are reported in this paper, in section VII-B.

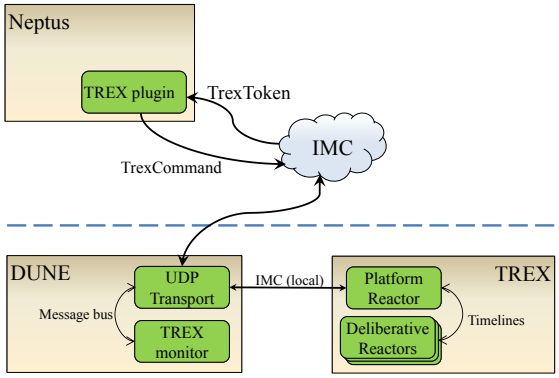

Fig. 9. Integration of TREX into the LSTS toolchain

A. IMC adaptations

IMC was adapted by adding two new messages:

- *TrexCommand* messages are used to send commands to TREX directly. These commands include enabling / disabling TREX control of the vehicle, posting new goals and recalling goals.
- *TrexToken* messages are generated in TREX reactors and are posted to the Neptus interface (and logged to disk) with all observation/plan tokens being produced by TREX. In TREX/EUROPA, a token can either be an observation that is derived from current and past states of the environment or may also be a an observation that should be made in the future (goal). All the tokens being produced by the reactors are this way logged and shared in real-time with Neptus if there is an available communication mean.

B. DUNE adaptations

In DUNE, we created a new DUNE monitor task that monitors the connection with the TREX process. The monitor is in normal state only if the TREX process is reporting a normal execution every 1 seconds.

C. Neptus adaptations

In Neptus, we created a new TREX plug-in that provides the following extensions:

1) A new map layer in the console that displays the state of all goals received by TREX. This plugin merges data from both the sent *TrexCommand* and *TrexToken* messages received from TREX.
2) A new map interaction mode adds the possibility (when active) to add/remove goals that are sent to TREX when the vehicle is connected.
3) A new MRA visualization was added to plot the timelines generated by TREX and logged as *TrexToken* messages.

D. TREX adaptations

In order to execute deliberative plans with TREX, we carefully described the domain model of our AUVs in NDDL [16] and created a *Platform* reactor that serves as a bridge between TREX and DUNE, translating DUNE vehicle state into TREX observations and transforming TREX goals into DUNE commands (IMC messages).

The platform reactor used DUNE as a library mostly for IMC communications. This reactor listens to IMC messages (coming from DUNE) and posts the corresponding observations to its controlled timelines. For instance, *EstimatedState* messages received from DUNE generate corresponding *Position* observations and *VehicleCommand* messages correspond to *Command* observations, etc.

The *command* timeline, controlled by the *Platform* reactor was special in the sense that it both posted the current maneuvers being executed by the vehicle but also accepted commands that would be translated into *VehicleCommand* IMC messages. *VehicleCommand* messages allow starting / stopping the execution of a maneuver onboard DUNE and this is how TREX could control the vehicle.

VII. Field Tests and Results

The LSTS toolchain has been tested numerous times in the field with hundreds of operational hours for both AUVs and UAVs. In this section we will report results of 2012 experiments.

A. Multi-vehicle experiments at Porto Harbor

In this experiment the vehicles LAUV-Xtreme-2 (*XT2*) and LAUV-Noptilus-1 (*NP1*) were used for testing interference of sonars when vehicles operate close to each other. *XT2* used LBL navigation while *NP1* used inertial navigation only. We did two separate tests:

1) Multi-vehicle survey: In this test, we had both vehicles executing a similar rows plan 20 meters apart from each other (see Fig. 10). The vehicles were always between 40 and 50 meters from each other and both registered sidescan imagery from the bottom while executing it. Moreover, *XT2* was also reporting its position using the acoustic modem every 60 seconds.

As a result it was possible to do a bathymetry and sidescan survey in half the time it would be possible using

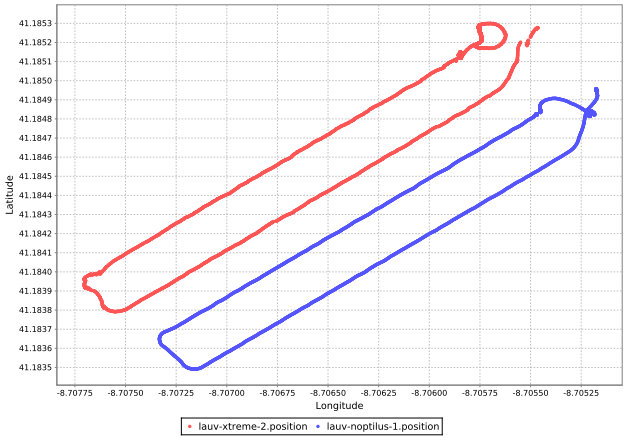

Fig. 10. Positions of vehicles in multi-vehicle survey test

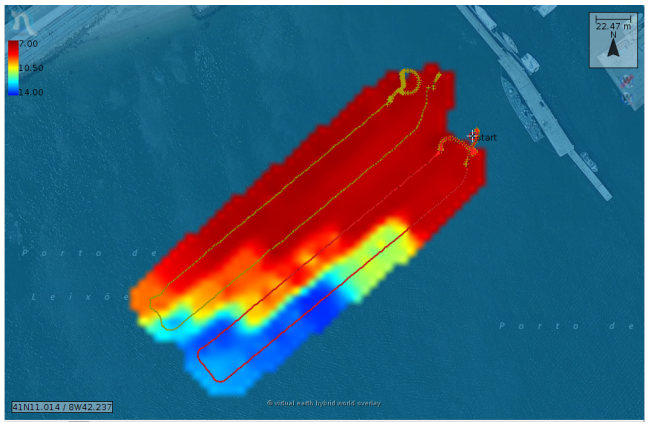

Fig. 11. Merged bathimetry data from multi-vehicle survey

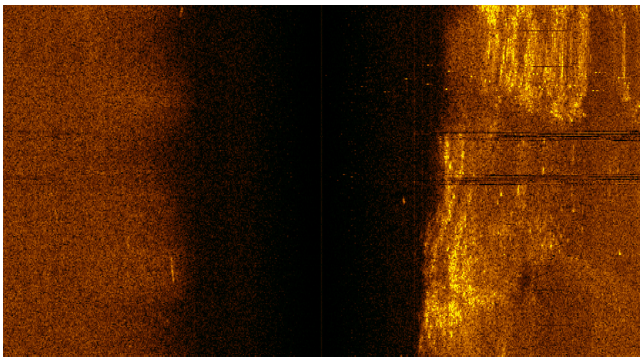

Fig. 12. External acoustic interferences in starboard transducer

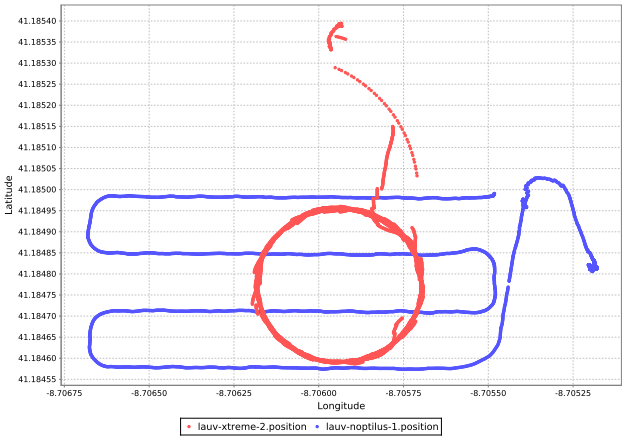

Fig. 13. Positions of vehicles in sidescan detection test

a single AUV (see Fig. 11). Also, in the *NP1* gathered sidescan data (Fig. 12) we can observe that there are external acoustic interferences coinciding with the times when *XT2* was sending its position using the acoustic modem. Also it is visible that this interference is more predominant in the starboard sidescan transducer than it is on the port transducer. This can be explained by of the location of *XT2* relative to *NP1*. We conclude that this interference was thus introduced solely by the position communications of the other vehicle (*XT2*).

2) Sidescan detection of a moving AUV: In this test, we had both vehicles operating in the same region, but at different depths. *XT2* was loitering in circles at 2 meters from the bottom while *NP1* was doing a survey at 1 meter from the surface (see Fig. 13).

The objective of this experiment was to test if it was feasible to detect *XT2* in the sidescan imagery of *NP1*. In the resulting data there is actually a sighting of a moving object near the bottom (see Fig. 14) exactly when both trajectories cross. The moving contact results in a line traced in the sidescan plot which moves out of the sidescan range after a few seconds. However this proves that sidescan imagery can be used to detect and possibly follow other UVs in the water.

B. Onboard deliberative planning

During the Rapid Environmental Picture event 2012 at Sesimbra [17], we field-tested the EUROPA deliberative planner [15] running onboard the LAUV-Seacon-1 (*SC1*) [18].

For this test, there was a TREX program running side-by-side with DUNE onboard *SC1*. On the TREX side, there was a reactor (module) that was capable of receiving high-level objectives from Neptus and forwarding them to deliberative reactors. These reactors, were then in charge of creating a plan that encompasses all the usual operational constraints (speed, depth, battery) as well as accomplish the desired objectives.

In this experiment, objectives were either points to be visited or areas to be surveyed. Moreover the vehicle was also required to surface periodically and obtain a GPS fix.

As expected, the planner rejected impossible objectives like the objectives outside the operational limits and never stayed underwater for more than the maximum allowed time. The plot in Fig. 15, shows the vehicle depth against number of visible GPS satellites as it was being controlled by the onboard deliberative planner. The maximum allowed time for the vehicle to be underwater in this test was 5 minutes.

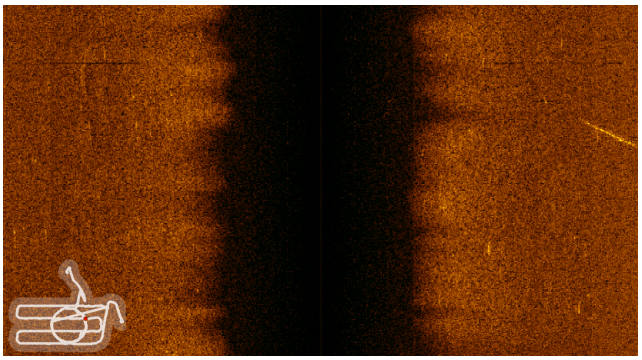

Fig. 14. Sidescan detection of moving AUV (trace on the right)

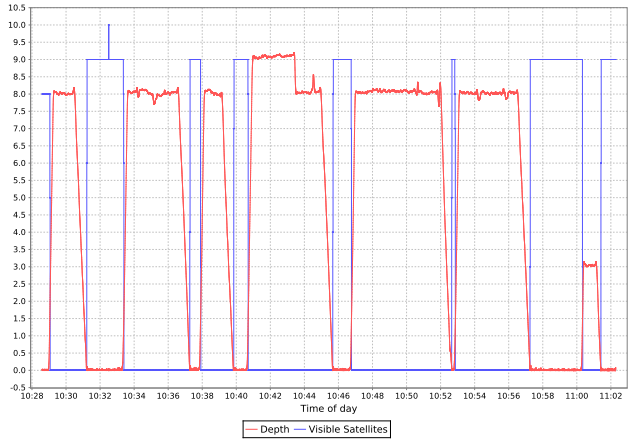

Fig. 15. Vehicle depth against visible GPS satellites during deliberatibe planning experiments

After this time, the planner should command the vehicle to obtain GPS fix, at the surface. This plot illustrates that the vehicle was capable of accomplishing more than one goals (at different depths) while underwater (10:40 - 10:46) and also it surfaced briefly before continuing the pursuit of objectives that took more than 5 minutes to conclude (10:48 - 10:58). When idle, the planner would come back to the surface in order to receive more goals from the user.

VIII. Conclusions and Future Work

We have described the overall architecture of our toolchain and showed how it can been used to control multiple unmanned vehicles of different types.

DUNE is currently running onboard all our vehicles and Neptus is used to control them. For this, the fact that we share the same IMC protocol among all components has proven to be the right decision concerning flexibility and adaptability of the framework to new objectives. This is especially noticeable in the documented extensions for MAVs and onboard deliberative planning.

Extending the LSTS toolchain for supporting new control algorithms, data visualizations or hardware is possible by implementing new tasks / plug-ins that are compatible with IMC and then all remaining toolchain modules will afterwards be compatible with it.

The current limitations of the toolchain towards the control of networked vehicle systems reside mostly on the complexity that operators still need to handle in order to be able to control several vehicles simultaneously. We aim to continue improve situational awareness and planning interfaces in an undergoing collaboration with the Monterey Bay Research Aquarium where we are trying to add onboard deliberative planners to all vehicles as well as at the operator consoles for aiding operators in the creation of plans and safe tasking of vehicles.

The toolchain has recently been open sourced and is now free to use and ready for contributions from other research groups. From this, we expect several new requirements, extensions and support for other currently unforeseen scenarios and results.

References

[1] R. Martins, P. Dias, J. Pinto, P. Sujit, and J. B. Sousa, *Multiple Underwater Vehicle Coordination for Ocean Exploration*. IJCAI, 2009.

[2] R. Martins, J. B. de Sousa, C. Carvalho Afonso, and M. Incze, "REP10 AUV: shallow water operations with heterogeneous autonomous vehicles," in *OCEANS, 2011 IEEE-Spain*. IEEE, 2011, pp. 1–6.

[3] R. Martins, "Disruption/delay tolerant networking with low-bandwidth underwater acoustic modems," in *Autonomous Underwater Vehicles (AUV), 2010 IEEE/OES*. IEEE, 2010, pp. 1–5.

[4] A. Tinka, S. Diemer, L. Madureira, E. Marques, J. B. Sousa, R. Martins, J. Pinto, J. E. Silva, A. Sousa, P. Saint-Pierre *et al.*, "Viability-based computation of spatially constrained minimum time trajectories for an autonomous underwater vehicle: implementation and experiments," in *American Control Conference, 2009. ACC'09.* IEEE, 2009, pp. 3603–3610.

[5] LSTS group on GitHub. [Online]. Available: https://github.com/LSTS

[6] R. Martins, P. S. Dias, E. R. Marques, J. Pinto, J. Sousa, and F. L. Pereira, "IMC: A communication protocol for networked vehicles and sensors," in *OCEANS 2009-EUROPE*. IEEE, 2009, pp. 1–6.

[7] IMC protocol specification and documentation. [Online]. Available: https://github.com/LSTS/imc

[8] DUNE source code repository. [Online]. Available: https://github.com/LSTS/dune

[9] Neptus source code repository. [Online]. Available: https://github.com/LSTS/neptus

[10] R. Gonçalves, P. S. Dias, J. Pinto, G. Gonçalves, and J. Sousa, "A STANAG 4586 compliant command and control operational interface for multiple UAVs," in *Humous 2010 Humans Operating Unmanned Systems, ISAE - Toulouse, ONERA - Toulouse, France*, 2010.

[11] NATO Standardization Agency (NSA), "STANAG 4586 ed. 2.5 feb 2007 draft," in *Standard Interfaces of UAV Control System (UCS) for NATO UAV Interoperability*. NATO, 2007.

[12] DIY Drones, "Ardupilot project," 2011.

[13] IGEPv2 processor board. [Online]. Available: http://www.isee.biz/products/processor-boards/igepv2-board

[14] F. Py, K. Rajan, and C. McGann, "A Systematic Agent Framework for Situated AutonomousSystems," in *9th International Conf. on Autonomous Agents and Multiagent Systems*, Toronto, Canada, May 2010.

[15] J. Bresina, A. K. Jónsson, P. H. Morris, and K. Rajan, "Mixed-initiative constraint-based activity planning for mars exploration rovers," in *Proceedings of 4th International Workshop on Planning and Scheduling for Space (IWPSS)*, 2004.

[16] NDDL reference manual. [Online]. Available: https://code.google.com/p/europa-pso/wiki/NDDLReference

[17] REP12 experiment web site. [Online]. Available: https://whale.fe.up.pt/rep12/

[18] J. Pinto, J. Sousa, F. Py, and K. Rajan, "Experiments with deliberative planning on autonomous underwater vehicles," in *IROS 2012 Workshop on Robotics for Environmental Monitoring, Vilamoura, Portugal*, 2012.