Enhanced Testing of Autonomous Underwater Vehicles using Augmented Reality & JavaBeans

Benjamin C. Davis and David M. Lane
Ocean Systems Laboratory, Heriot-Watt University
Scotland

1. Introduction

System integration and validation of embedded technologies has always been a challenge, particularly in the case of autonomous underwater vehicles (AUVs). The inaccessibility of the remote environment combined with the cost of field operations have been the main obstacles to the maturity and evolution of underwater technologies. Additionally, the analysis of embedded technologies is hampered by data processing and analysis time lags, due to low bandwidth data communications with the underwater platform. This makes real-world monitoring and testing challenging for the developer/operator as they are unable to react quickly or in real-time to the remote platform stimuli.

This chapter discusses the different testing techniques useful for unmanned underwater vehicle (UUVs) and gives example applications where necessary. Later sections digress into more detail about a new novel framework called the Augmented Reality Framework (ARF) and its applications on improving pre-real-world testing facilities for UUVs. To begin with more background is given on current testing techniques and their uses. To begin with some background is given about Autonomous Underwater Vehicles (AUVs).

An AUV (Healey et al., 1995) is a type of UUV. The difference between AUVs and Remotely operated vehicles (ROVs) is that AUVs employ intelligence, such as sensing and automatic decision making, allowing them to perform tasks autonomously, whilst ROVs are controlled remotely by a human with communications running down a tether. AUVs can operate for long periods of time without communication with an operator as they run a predefined mission plan. An operator can design missions for multiple AUVs and monitor their progress in parallel. ROVs require at least one pilot per ROV controlling them continuously. The cost of using AUVs should be drastically reduced compared with ROVs providing the AUV technology is mature enough to execute the task as well as an ROV. AUVs have no tether, or physical connection with surface vessels, and therefore are free to move without restriction around or inside complex structures. AUVs can be smaller and have lower powered thrusters than ROVs because they do not have to drag a tether behind them. Tethers can be thousands of metres in length for deep sea missions and consequently very heavy. In general, AUVs require less infrastructure than ROVs i.e. ROVs usually require a large ship and crew to operate which is not required with an AUV due to being easier to deploy and recover.

In general, autonomous vehicles (Zyda et al., 1990) can go where humans cannot, do not want to, or in more relaxed terms they are suited to doing the "the dull, the dirty, and the

Source: Underwater Vehicles, Book edited by: Alexander V. Inzartsev,
ISBN 978-953-7619-49-7, pp. 582, December 2008, I-Tech, Vienna, Austria

dangerous”. One of the main driving forces behind AUV development is automating , potentially tedious, tasks which take a long time to do manually and therefore incur large expenses. These can include oceanographic surveys, oil/gas pipeline inspection, cable inspection and clearing of underwater mine fields. These tasks can be monotonous for humans and can also require expensive ROV pilot skills. AUVs are well suited to labour intensive or repetitive tasks, and can perform their jobs faster and with higher accuracy than humans. The ability to venture into hostile or contaminated environments is something which makes AUVs particularly useful and cost efficient.

AUVs highlight a more specific problem. Underwater vehicles are expensive because they have to cope with the incredibly high pressures of the deepest oceans (the pressure increases by 1 atmosphere every 10m). The underwater environment itself is both hazardous and inaccessible which increases the costs of operations due to the necessary safety precautions. Therefore the cost of real-world testing, the later phase of the testing cycle, is particularly expensive in the case of UUVs. Couple this with poor communications with the remote platform (due to slow acoustic methods) and debugging becomes very difficult and time consuming. This incurs huge expenses, or more likely, places large constraints on the amount of real-world testing that can be feasibly done. It is paramount that for environments which are hazardous/inaccessible, such as sea, air and space, that large amounts of unnecessary real-world testing be avoided at all costs. Ideally, mixed reality testing facilities should be available for pre-real-world testing of the platform. However, due to the expense of creating specific virtual reality testing facilities themselves, adequate pre-real-world tests are not always carried out. This leads to failed projects crippled by costs, or worse, a system which is unreliable due to inadequate testing.

Different testing mechanisms can be used to keep real-world testing to a minimum. Hardware-in-the-loop (HIL), Hybrid Simulation (HS) and Pure Simulation (PS) are common pre-real-world testing methods. However, the testing harness created is usually very specific to the platform. This creates a problem when the user requires testing of multiple heterogeneous platforms in heterogeneous environments. Normally this requires many specific test harnesses, but creating them is often time consuming and expensive. Therefore, large amounts of integration tests are left until real-world trials, which is less than ideal.

Real world testing is not always feasible due to the high cost involved. It would be beneficial to test the systems in a laboratory first. One method of doing this is via pure simulation (PS) of data for each of the platform’s systems. This is not a very realistic scenario as it doesn’t test the actual system as a whole and only focuses on individual systems within a vehicle. The problem with PS alone is that system integration errors can go undetected until later stages of development, since this is when different modules will be tested working together. This can lead to problems later in the testing cycle by which time they are harder to detect and more costly to rectify. Therefore, as many tests as possible should to be done in a laboratory. A thorough testing cycle for a remote platform would include HIL, HS and PS testing scenarios. For example, an intuitive testing harness for HIL or HS would include: A 3D Virtual world with customisable geometry and terrain allowing for operator observation; A Sensor simulation suite providing exterioceptive sensor data which mimics the real world data interpreted by higher level systems; and a distributed communication protocol to allow for swapping of real for simulated systems running in different locations.

Thorough testing of the remote platform is usually left until later stages of development because creating a test harness for every platform can be complicated and costly. Therefore, when considering a testing harness it is important that it is re-configurable and very generic in order to accommodate all required testing scenarios. The ability to extend the testing harness to use specialised modules is important so that it can be used to test specialized systems. Therefore a dynamic, extendible testing framework is required that allows the user to create modules in order to produce the testing scenario quickly and easily for their intended platform/environment.

2. Methods of testing

Milgrim's Reality-Virtuality continuum (Takemura et al., 1994), shown in Figure 1, depicts the continuum from reality to virtual reality and all the hybrid stages in between. The hybrid stages between real and virtual are known as augmented reality (Behringer et al., 2001) and augmented virtuality. The hybrid reality concepts are built upon by the ideas of Hardware-in-the-loop (HIL) and Hybrid Simulation (HS). Figure 1 shows how the different types of testing conform to the different types of mixed reality in the continuum. There are 4 different testing types:

1. **Pure Simulation** (PS) (Ridao et al., 2004) - testing of a platform's modules on an individual basis before being integrated onto the platform with other modules.
2. **Hardware-in-the-loop** (HIL) (Lane et al, 2001) - testing of the real integrated platform is carried out in a laboratory environment. Exterioceptive sensors such as sonar or video, which interact with the intended environment, may have to be simulated to fool the robot into thinking it is in the real world. This is very useful for integration testing as the entire system can be tested as a whole allowing for any system integration errors to be detected in advance of real world trials.
3. **Hybrid Simulation** (HS) (Ridao et al., 2004; Choi & Yuh, 2001) - testing the platform in its intended environment in conjunction with some simulated sensors driven from a virtual environment. For example, virtual objects can be added to the real world and the exterioceptive sensor data altered so that the robot thinks that something in the sensor dataset is real. This type of system is used if some higher level modules are not yet reliable enough to be trusted to behave as intended using real data. Consequently, fictitious data is used instead, augmented with the real data, and inputted to the higher level systems. Thus, if a mistake is made it doesn't damage the platform. An example of this is discussed in section 4.2.
4. **Real world testing** - This is the last stage of testing. When all systems are trusted the platform is ready for testing in the intended environment. All implementation errors should have been fixed in the previous stages otherwise this stage is very costly. For this stage to be as useful as possible the system designers and programmers need to have reliable intuitive feedback, in a virtual environment, about what the platform is doing otherwise problems can be very hard to see and diagnose.

ARF provides functionality across all stages of the continuum allowing for virtually any testing scenario to be realised. For this reason it is referred to as a mixed reality framework.

In the case of Augmented Reality, simulated data is added to the real world perception of some entity. For example, sonar data on an AUV could be altered so that it contains fictitious objects i.e. objects which are not present in the real world, but which are present in the virtual world. This can be used to test the higher level systems of an AUV such as obstacle detection (See *Obstacle detection and avoidance example in Section 4.2).* A virtual world is used to generate synthetic sensor data which is then mixed with the real world data. The virtual world has to be kept in precise synchronization with the real world. This is commonly known in Augmented Reality as the *registration* problem. The accuracy of registration is dependent on the accuracy of the position/navigation systems onboard the platform. Registration is a well known problem with underwater vehicles when trying to match different sensor datasets to one another for visualisation. Accurate registration is paramount for displaying the virtual objects in the correct position in the simulated sensor data.

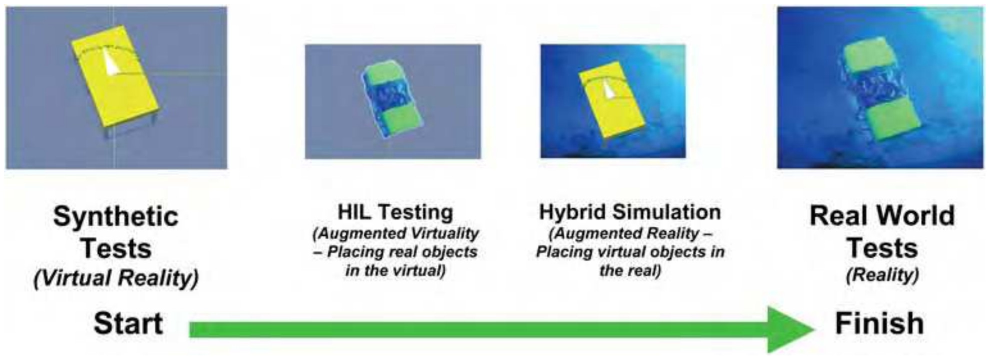

Synthetic Tests (Virtual Reality)
HIL Testing (Augmented Virtuality – Placing real objects in the virtual)
Hybrid Simulation (Augmented Reality – Placing virtual objects in the real)
Real World Tests (Reality)
Start
Finish

Fig. 1. Reality Continuum combined with Testing Types.

Augmented Virtuality is the opposite of augmented reality i.e. instead of being from a robot's/person's perspective it is from the virtual world's perspective - the virtual world is augmented with real world data. For example, real data collected by an AUV's sensors is rendered in real time in the virtual world in order to recreate the real world in virtual reality. This can be used for *Online Monitoring* (OM) and *operator training* (TR) (Ridao et al., 2004). This allows an AUV/ROV operator to see how the platform is situated in the remote environment, thus increasing situational awareness.

In Hybrid Simulation the platform operates in the real environment in conjunction with some sensors being simulated in real time by a synchronized virtual environment. Similar to Augmented Reality, the virtual environment is kept in synchronization using position data transmitted from the remote platform. Thus simulated sensors are attached to the virtual platform and moved around in synchronization with the real platform. Simulated sensors collect data from the virtual world and transmit the data back to the real systems on the remote platform. The real systems then interpret this data as if it were real. It is important that simulated data is very similar to the real data so that the higher level systems cannot distinguish between the two. In summary, the real platform's perception of the real environment is being augmented with virtual data. Hence HS is inherently Augmented Reality. An example of a real scenario where AR testing procedures are useful is in obstacle detection and avoidance in the underwater environment by an AUV. See *Obstacle detection and avoidance example in* Section 4.2.

Hardware-in-the-Loop (HIL) is another type of mixed reality testing technique. This type of testing allows the platform to be tested in a laboratory instead of in its intended environment. This is achieved by simulating all required exterioceptive sensors using a virtual environment. Virtual sensor data is then sent to the real platform's systems in order to fool them. In essence this is simply virtual reality for robots. Concurrently, the outputs of

higher level systems, which receive the simulated data, can be relayed back and displayed in the virtual environment for operator feedback. This can help show the system developer that the robot is interpreting the simulated sensor data correctly. HIL requires that all sensors and systems that interact directly with the virtual environment are simulated. Vehicle navigation systems are a good example since these use exterioceptive sensors, actuators and motors to determine position. Using simulated sensors means that the developer can specify exactly the data which will be fed into the systems being tested. This is complicated to do reliably in the real environment as there are too many external factors which cannot be easily controlled. Augmenting the virtual environment with feedback data from platform for observation means that HIL can be Augmented Virtuality as well as merely virtual reality for the platform.
Consequently, HIL and HS are both deemed to be Mixed Reality concepts, thus any testing architecture for creating the testing facilities should provide all types of mixed reality capabilities and be inherently distributed in nature.

3. ARF

The problem is not providing testing facilities as such, but rather being able to create them in a timely manner so that the costs do not outweigh the benefits. Any architecture for creating mixed reality testing scenarios should be easily configurable, extendable and unrestrictive so that it is feasible to create the testing facilities rather than do more expensive and less efficient real world tests. In essence, creating testing facilities requires a short term payout for a long term gain. Long term gains only applicable if the facilities are extendable and re-configurable for different tasks.
ARF is a component based architecture which provides a framework of components that are specifically designed to facilitate the rapid construction of mixed reality testing facilities. ARF provides a generic, extendable architecture based on JavaBeans and Java3D by Sun Microsystems. ARF makes use of visual programming and introspection techniques, to infer information about components, and consequently provides users with help via guided construction for creating testing scenarios. This allows for rapid prototyping of multiple testing combinations allowing virtual reality scenarios to be realised quickly for a multitude of different applications.
There are other architectures which provide HIL, HS and PS capabilities such as Neptune (Ridao et al., 2004). However, they only focus on testing and do not provide the extendibility and low level architecture that allows for easy extension using users own components. This is where ARF provides enhanced capabilities since it uses visual programming to provide a more intuitive interface allowing quick scenario creation and for configurations to be changed quickly and easily by abstracting the user from modifying the configuration files directly.

3.1 Visual programming

A subject that has been touched upon is that of visual programming or guided construction. One of the main performance inhibiting problems for software programmers and designers is whether or not they know exactly how all of the software modules they require work, and knowing how to use them. More often than not computer software modules are poorly documented and do not provide example implementations on how to use them. This is particularly the case when projects are on a tight schedule with little capital backing them, since there simply isn't the time or the money to spend on creating nicely commented and documented code. This problem is self perpetuating since each time a badly documented module is required, the programmer spends so much time figuring out how to use the module that they then have less time to document their own code.

Poor documentation is merely one aspect which decreases productivity when it comes to developing software modules. Another problem for the programmer is knowing which modules are available and their functionality. Quite often package names and module names are not enough for the programmer to determine a module's functionality. Consequently, the programmer has to trawl through the API specification for the entire programming library to find out whether or not it is useful to them. Documentation maybe poor or non-existent, even if it does exist it can be time consuming to find out exactly what to do to use the module because no sensible examples are given. Thus, most of the time spent by the programmer is not spent actually programming. Conversely, when a programmer knows exactly the functionality of a module they can create a program to use it with great speed. Therefore, any architecture which reduces the amount of time spent by the programmer looking at documentation the faster they can finish the task.

This combination of problems means that programmers spend a lot of time re-inventing the wheel since existing modules are hard to locate, poorly documented or impossible to use. This problem is rife when it comes to producing virtual environments and simulated modules for testing robots, especially AUVs. This is usually due to environments being quickly “hacked up” to fulfil one purpose without considering the many other potential usages. Monolithic programming approaches then make reconfiguration and extension almost impossible. Add-ons can sometimes be “hacked” into the existing program, but in essence it is still a very inflexible program that will eventually become obsolete because a new usage or platform is required. At this stage it may be too complicated to try and extend the existing program to fulfil the new requirements, so instead a new program is quickly created with a few differences but is in essence just as inflexible as the first. More time spent making generic modules with basic inputs and basic outputs in a configurable environment makes making changes later on quicker and easier. However, when completely new functionality is required, a configurable environment still has to be reprogrammed to incorporate new modules. This can be difficult unless the environment is specifically designed to allow extension.

Visual programming (Hirakawa & Ichikawa, 1992) provides a solution for rapid module development and provides some ideas which can be harnessed to provide the basic idea behind a generic architecture for creating virtual environments. Visual programming is the activity of making programs through spatial manipulations of visual elements. It is not new and has been around since the very early 1970s when logic circuits were starting to be designed using Computer Aided Design (CAD) packages. Visual programming is more intuitive than standard computer programming because visual programming provides more direct communication between human and computer which given the correct visual queues makes it easier for the user to understand the relationships between entities, and thus makes connecting components easier. Consider the example of taking ice cube trays out of the freezer and placing one in a drink. The human way of doing this is simply to look to locate the ice cubes, use the hands to manipulate the ice cube out of the tray and then drop it into the drink. Thus any interface which allows the user to work in their natural way is going to make the job quicker. The other option, which is more like computer programming, is to have the human write down every single action required to do this. The visual programming approach might be to manipulate a visual representation of the ice cube trays by dragging and clicking a mouse. Programming is far more clunky and will take much longer as it is not as intuitive. Therefore, visual programming aims to exploit natural human instincts in order to be as intuitive and effective as possible.

3.2 JavaBeans

Visual programming is a good idea, however, due to its visual nature it places requirements on how a module is written. This usually requires that the low level components be programmed in a specially designed language which provides more information to the visual programming interface. This leads to visual programming only being used for more specific uses, such as connecting data flows using CAD packages. However, visual programming can become far more powerful if it places nearly zero restrictions on how the low level components are created i.e. the programming language used. In order for visual programming to be widely accepted it has to some how make use of existing software components even if they are not designed to be used in this way. One such method of visual programming exists whereby components only have to implement a few simple “programming conventions” in order to be able to be used visually. These special software components are called JavaBeans and are based on the Java programming language (Sun Microsystems).

JavaBean visual programming tools work on the basis that a Java class object has been programmed adhering to certain coding conventions. Using this assumption the visual programming tool is able to use introspection techniques to infer what the inputs and outputs to a Java class and then display these as properties to the user. Thanks to Java's relatively high level byte code compilation layer, it is relatively simple for a JavaBean processor to analyse any given class and produce a set of properties which a user can edit visually. Therefore, removing the need for the programmer to write code in order to allow the configuration of Java class objects.

JavaBean programming environments currently exist which allow a user to connect and configure JavaBeans to make 2D GUI based applications. The BeanBuilder (https://bean-builder.dev.java.net) is one such program which provides the user with an intuitive visual interface for creating software out of JavaBeans. However, this doesn’t provide any extra guidance other than graphical property sheet generation. A virtual environment is needed for mixed reality testing scenarios and this cannot be easily provided using the BeanBuilder’s current platform. However, JavaBeans offer a very flexible base upon which a virtual environment development tool can be built. Since it can easily be extended via JavaBeans and all the advantages of JavaBeans can be harnessed. Another advantage of using JavaBeans is that scenario configurations can be exported to XML file for distribution to others and manual configuration.

3.3 Architecture

ARF provides the ability to execute all testing regimes across the reality continuum. It does this by incorporating: the OceanSHELL distributed communication protocol, vehicle dynamics & navigation simulators, sensor simulation, an interactive three-dimensional (3d) virtual world and information display. All spatially distributed components are easily interconnected using message passing via the communication protocol, or directly by method call using ARF’s visual programming interface based on JavaBeans. The key to ARF’s HIL and HS capabilities is the flexibility of the communications protocol. Other

external communications protocols are easily implemented by extending the event passing currently used by ARF's JavaBeans.
ARF provides a new type of JavaBean which allows the user to create a 3D environment out of JavaBeans. It is called a Java3DBean and is based on Java3D and JavaBeans. Java3DBeans inherently have all the functionality of JavaBean objects but with the added advantage that they are Java3D Scenegraph nodes. This gives them extra features and functionality such as 3D geometry, behaviours and able to interact with other Java3DBeans within the virtual world. ARF provides a user interface which extends the JavaBean PropertySheet allowing for Java3DBeans to be configured in the same way. The user is able to construct the 3D environment using Java3DBeans and decide which data to input/output to/from the real world. This provides unlimited functionality for HIL, HS and PS testing since any communication protocol can be implemented in JavaBeans and used to communicate to/from ARF to a remote platform. Mixed reality techniques can be used to render data visually in order to increase the situational awareness of an operator of a UUV, and provide simulation of systems for testing the remote platform. This increases the rate at which errors are detected, resulting in a more robust system in less time.

3.4 OceanSHELL distributed communications protocol

The obstacle detection and avoidance example *(see section 4.2)* highlights the need for a location transparent communication system. ARF requires real modules to be able to be swapped for similar simulated modules without the other systems knowing, having to be informed or programmed to allow it. The underlying communication protocol which provides the flexibility needed by the framework is OceanSHELL (Ocean Systems Lab, 2008). OceanSHELL provides distributed communications via UDP packets allowing modules to run anywhere i.e. provides module location transparency. Location transparency makes mixed reality testing straight forward because modules can run either on the remote platform, or somewhere else such as a laboratory.
OceanSHELL is a software library implementing a low overhead architecture for organising and communicating between distributed processes. OceanSHELL's low overhead in terms of execution speed, size and complexity make it eminently suited for embedded applications. An extension to OceanShell, called JavaShell, is portable because it runs on Java platforms. Both JavaShell and OceanShell fully interact, the only difference being that OceanShell uses C structures to specify message definitions instead of the XML files which JavaShell uses. However, both systems are fully compatible. OceanShell is not only platform independent but also language independent, increasing portability.
ARF allows for dynamic switching of OceanSHELL message queues and changing of port number. This allows for information flows to be re-routed by simulated modules in real time. This is ideal for doing HS or HIL testing. Figure 2 shows how OceanSHELL is used to communicate between the remote environment and the virtual environment.

3.5 ARF features

The Augmented Reality Framework (ARF) is a configurable and extendible virtual reality framework of tools for creating mixed reality environments. It provides sensor simulation, sensor data interpretation, visualisation and operator interaction with the remote platform. ARF can be extended to use many sensors and data interpreters specific to the needs of the user and target domain. ARF is also domain independent and can be tailored to the specific needs of the application. ARF provides modularity and extendibility by providing

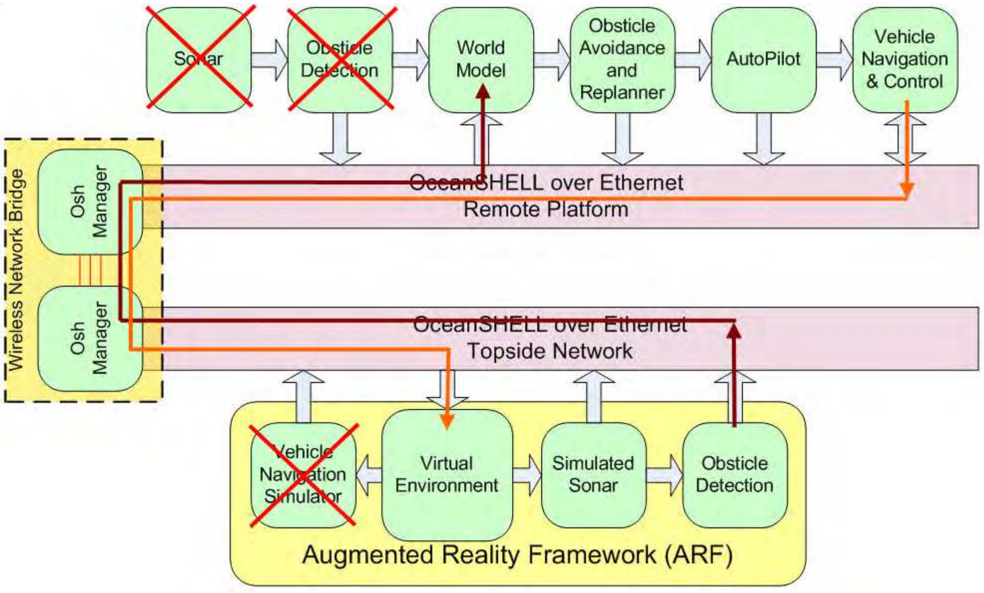


Fig. 2. This diagram shows how OceanSHELL provides the backbone for switching between real and simulated (topside) components for use with HS/HIL.

mechanisms to load specific modules created by the user, and provides a visual programming interface used to link together the different components. Figure 3 shows the ARF graphical user interface which the user uses to create their virtual environment. The 3D virtual environment is built using the scenegraph displayed in the top left of figure 3.

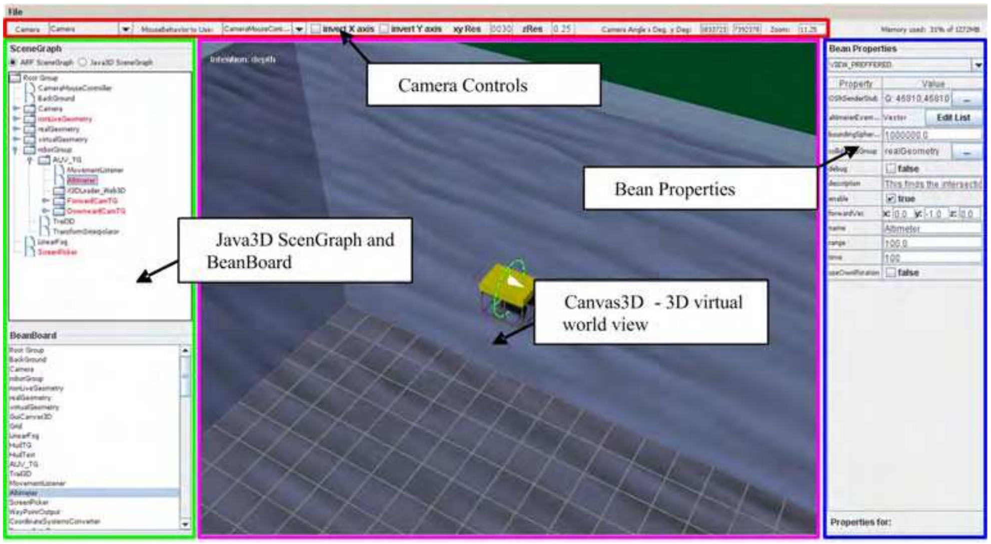


Fig. 3. ARF GUI Overview

ARF provides many programming libraries which allow a developer to create their own components. In addition ARF has many components ready for the user to create their own tailored virtual environment. The more ARF is used, the more the component library will grow, providing greater and greater flexibility, therefore exponentially reducing scenario creation times.

The ARF framework provides a 3D virtual world which Java3DBeans can use to display data and to sense the virtual environment. ARF provides many basic components to build virtual environments from. These components can then be configured specifically to work as desired by the user. If the required functionality does not exist the user can programme their own components and add them to the ARF component library. For example, a component could be a data listener which listens for certain data messages from some sensor, on some communication protocol (OceanSHELL, serial etc), and then displays the data "live" in the virtual environment. The component may literally be an interface to a communications protocol like OceanSHELL, from which other components can be connected in order to transmit and receive data.

ARF has the ability to create groups of configured components which perform some specific task or make up some functional unit. This group of components can then be exported as a super-component to the ARF component library for others to use. For example, an AUV super-component could include: a 3D model of an AUV, Vehicle Dynamics simulator, sonar, and a control input to the Vehicle Dynamics (keyboard or joystick). These virtual components can then be substituted for the real AUV systems for use with HIL and HS or copied and pasted to provide multiple vehicle support.

ARF allows complete scenarios to be loaded and saved so that no work is required to recreate an environment. ARF has components which provide interfaces to OceanSHELL, sensor simulation (sonar and video) and provides components for interpreting live OceanSHELL traffic and displaying it meaningfully in the virtual world. Figures 7 & 8 show a simple Sonar simulation using the ARF virtual environment. This sonar can then be output to a real vehicle's systems for some usage i.e. obstacle detection.

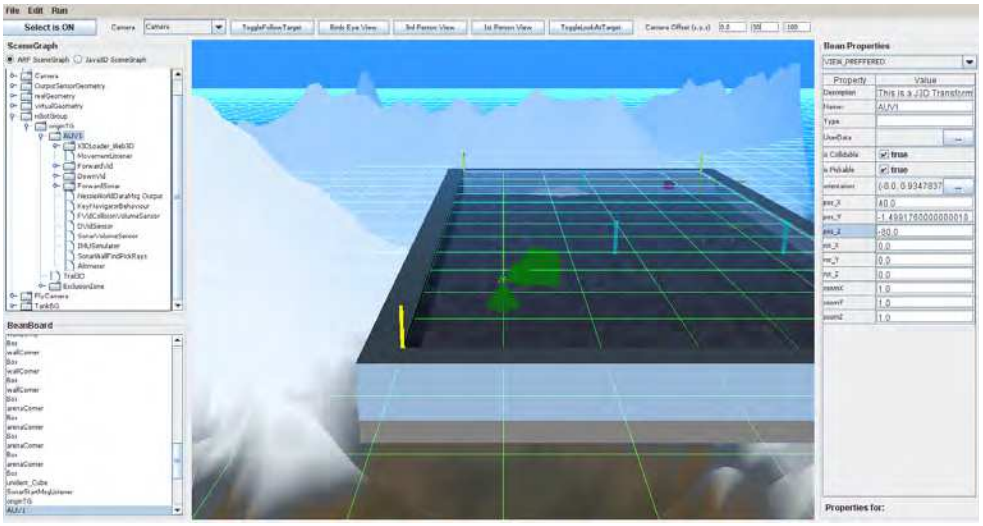


Fig. 4. Sauc-e AUV Competition Virtual environment for HIL testing 2007.

ARF provides a collection of special utility components for the user. These components are also provided in the programming API to be extended by the programmer to create their own JavaBeans. Java3DBeans are merely extensions to Java3D objects which adhere to the JavaBean programming conventions. ARF is capable of identifying which Beans are Java3DBeans and therefore knows how to deal with them. The only real difference between Java3DBeans and JavaBeans is that Java3DBeans are added to the 3D virtual world part of ARF as JavaBeans are only added as objects to the ARF BeanBoard (which keeps track of all objects). However, Java3DBeans can still communicate with any other objects in ARF's BeanBoard in the same way as JavaBeans.

In summary, JavaBeans are a collection of conventions which, if the programmer adheres to, allow a Java class to be dynamically loaded and configured using a graphical interface. The configurations of objects can also be loaded and saved at the click of a mouse button to a simple human readable XML file.

ARF provides many utility JavaBeans and Java3DBeans which the user can use directly, or extend. These include:

- Geometric Shapes for building scenes
- Mesh File loaders for importing VRML, X3D, DXF and many more 3D file types.
- Input listeners for controlling 3d objects with input devices (keyboard, mouse, joystick).
- Behaviours for making 3d objects do things such as animations.
- Camera control for inspecting and following the progress of objects.
- OceanSHELL input/output behaviours for rendering real data and for outputting virtual data from simulated sensors.
- Basic sensors for underwater technologies are provided such as forward looking sonar, sidescan sonar, bathymetric sonar, altimeter, inertial measurement unit (IMU), Doppler velocity log (DVL) etc.
- Vehicle Dynamics Models for movement simulation.

4. Applications of ARF

It is very hard to measure the effectiveness of ARF in improving the performance of creating testing scenarios. Performance Testing (see section 5) alone does not reflect how ARF is likely to be used and also does not demonstrate ARF's flexibility either. Although the potential applications are innumerable, this section describes some representative examples of applications and topics of research that are already gaining benefits from the capabilities provided by the Augmented Reality Framework.

4.1 Multiple vehicle applications

The main objective of the European Project GREX (http://www.grex-project.eu) is to create a conceptual framework and middleware systems to coordinate a swarm of diverse, heterogeneous physical objects (underwater vehicles) working in cooperation to achieve a well defined practical goal (e.g. search of hydrothermal vents) in an optimised manner.

In the context of GREX, algorithms for coordinated control are being developed. As these algorithms need to be tested on different vehicle platforms (and for different scenarios), real testing becomes difficult due to the cost of transporting and using vehicles; furthermore, the efficiency and safety of the different control strategies needs to be tested. The ARF virtual environment provides the ideal test bed: simulations can be run externally and fed into the virtual AUVs, so that the suitability of the different control strategies can be observed. The virtual environment serves not only as an observation platform, but can be used to simulate sensors for finding mines as used in the DELPHÍS multi-agent architecture (Sotzing et al., 2007), depicted in Figure 6.

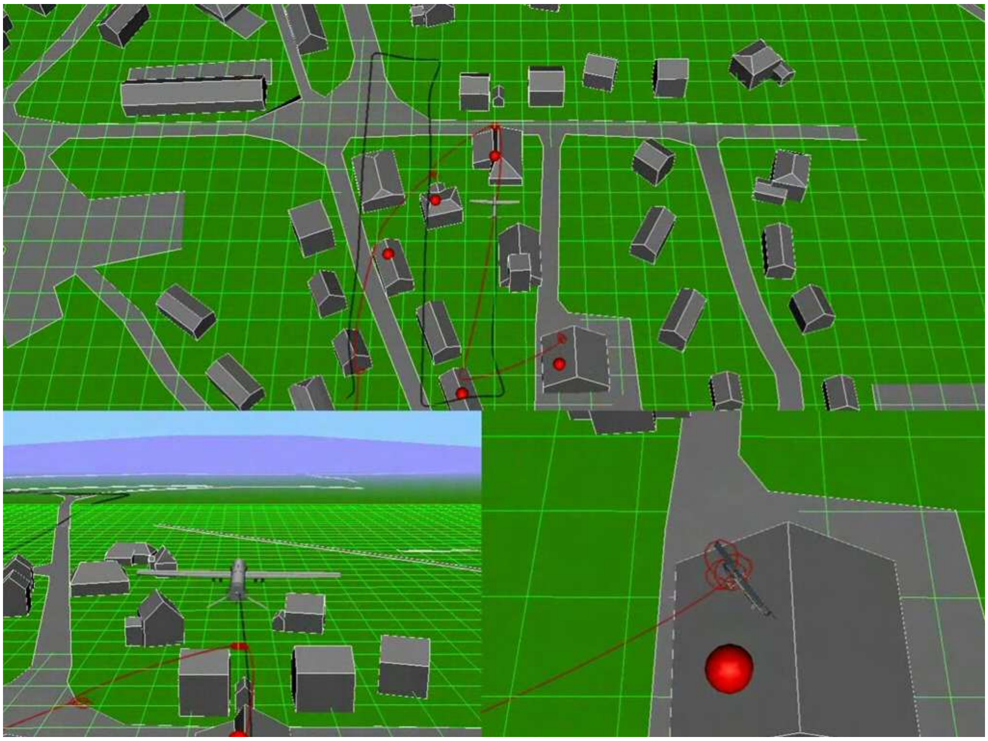


Fig. 5. Simulated UAVs cooperating and collaborating to complete a mission more efficiently.

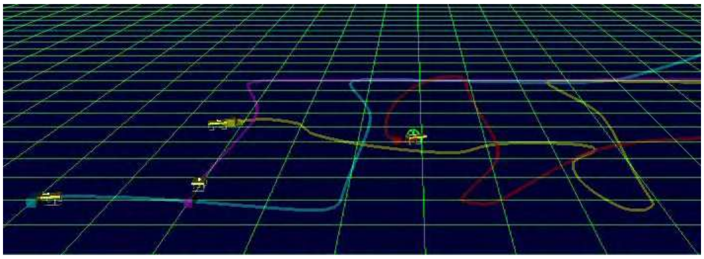


Fig. 6. Multiple AUVs running completely synthetically cooperating and collaborating to complete a mission more efficiently.

Other applications of the DELPHÍS multi agent architecture have been demonstrated using ARF. These include a potential scenario for the MOD Grand Challenge. This involves both surface and air vehicles working together to find targets, in a village, and inspect them. DELPHÍS was tested using simulated air vehicles, rather than underwater vehicles, which executed a search, classify & inspection task. ARF provided the virtual environment, vehicle simulation and object detection sensors required for the identification of potential threats in the scenario. Figure 5 displays the virtual environment view of the Grand Challenge scenario. The top of the screen shows a bird's eye observation of the area with the bottom left & right views following the survey class unmanned aerial vehicle (UAV) and inspection class UAV respectively. The red circles represent targets which the survey class UAV will detect upon coming within range of the UAV's object sensor. Information regarding specific targets of interest is shared between agents utilising the DELPHÍS system. Vehicles with the required capabilities can opt to further investigate the detected objects and reserve that task from being executed by another agent. Figure 6 shows AUVs working together to complete a lawn mower survey task of the sea bottom. The DELPHÍS system executes this quicker than a single AUV since each AUV agent does a different lawn mower leg. The agents appear to use a divide and conquer method, however, this is achieved completely autonomously and is not pre-programmed. The AUVs decide which lawn mower legs to execute based on distance to leg, predictions as to what other agents will choose, and which tasks have already been reserved by other competing self interested agents.

Apart from ARF being used for observation purposes, multiple AUVs/UAVs were simulated which helped tremendously for observing behaviours and testing of the DELPHÍS architecture. Basic object detection sensors provided a simple but effective method of outputting data from the virtual environment to DELPHÍS. Detections of certain types of objects meant that new goals were added to the plan. Generally these would be of the form: identify an object with one vehicle, then classify that object with another type of vehicle with the appropriate sensor. Thus some of the simulated AUVs were only capable of detecting the objects, whilst others were capable of inspecting and classifying those objects.

4.2 Obstacle detection and avoidance

One of the most common problems for unmanned vehicles is trajectory planning. This is the need to navigate in unknown environments, trying to reach a goal or target, while avoiding obstacles. These environments are expected to be in permanent change. As a consequence, sensors are installed on the vehicle to continuously provide local information about these changes. When object detections or modifications are sensed, the platform is expected to be able to react in real time and continuously adapt it's trajectory to the current mission targeted waypoint.

Testing these kinds of adaptive algorithms requires driving the vehicle against man-made structures in order to analyse its response behaviours. This incurs high collision risks on the platform and clearly compromises the vehicle's survivability.

A novel approach to this problem uses ARF to remove the collision risk during the development process. Using Hybrid Simulation, the approach uses a set of simulated sensors for rendering synthetic acoustic images from virtually placed obstacles. The algorithms are then debugged on a real platform performing avoidance manoeuvres over the virtual obstacles in a real environment. Figure 2 shows the required framework components, Figure 7 shows the virtual environment view and Figure 8 shows the resulting simulated sonar of the obstacles. It should be noted that the topside simulated components can be switched on to replace the remote platform's real components, therefore achieving HIL or HS. A detailed description of the evaluation and testing of obstacle avoidance algorithms for AUVs can be found in Pêtrès et al. (2007) & Patrón et al. (2005).

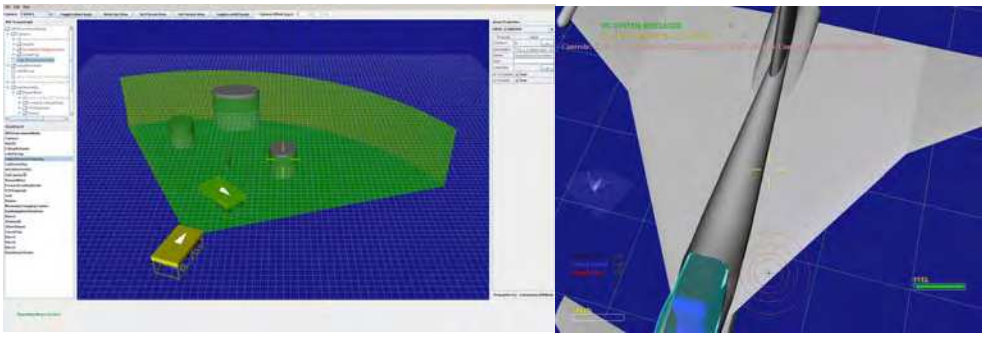


Fig. 7. ARF simulating Forward-look sonar of virtual objects.

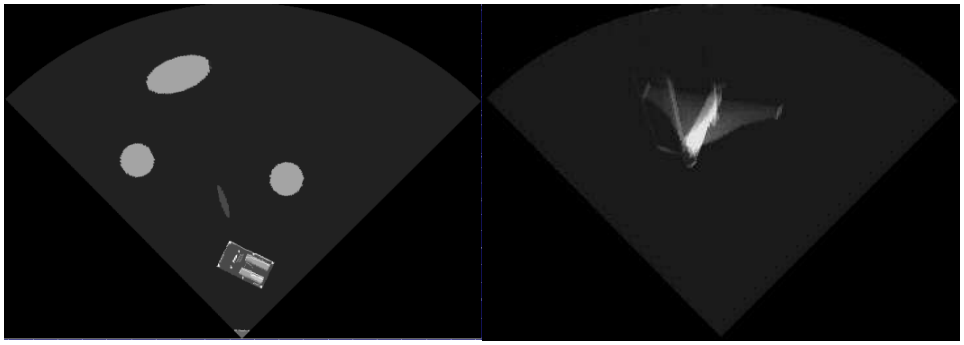


Fig. 8. The resulting images of the simulated Forward look sonar.

4.3 Autonomous tracking for pipeline inspection

Oil companies are raising their interest in AUV technologies for improving large field oil availability and, therefore, production. It is known that Inspection, Repair and Maintenance (IRM) comprise up to 90% of the related field activity. This inspection is clearly dictated by the vessels availability. One analysis of potential cost savings is using an inspection AUV. The predicted savings of this over traditional methods for inspecting a pipeline network system are up to 30%.
Planning and control vehicle payloads, such as the AUTOTRACKER payload (Patrón et al., 2006), can provide such capabilities. However, as mentioned, vessel availability and off-

shore operation costs make these types of payloads a difficult technology to evaluate. ARF can provide simulated sidescan sonar sensors for synthetic generated pipeline rendering. These capabilities provide a transparent interface for the correct and low cost debug of the tracking technologies. Furthermore, potentially complicated scenarios, such as multiple pipeline tracking and junctions of pipes, can be easily created to test the pipe detection and decision algorithms of the AUTOTRACKER system (see figure 9). This could not easily be tested in the real environment as real-time debugging is not available and the potential for incorrect decision due to confusion about which pipeline to follow is high, therefore higher risk of loss of an AUV.

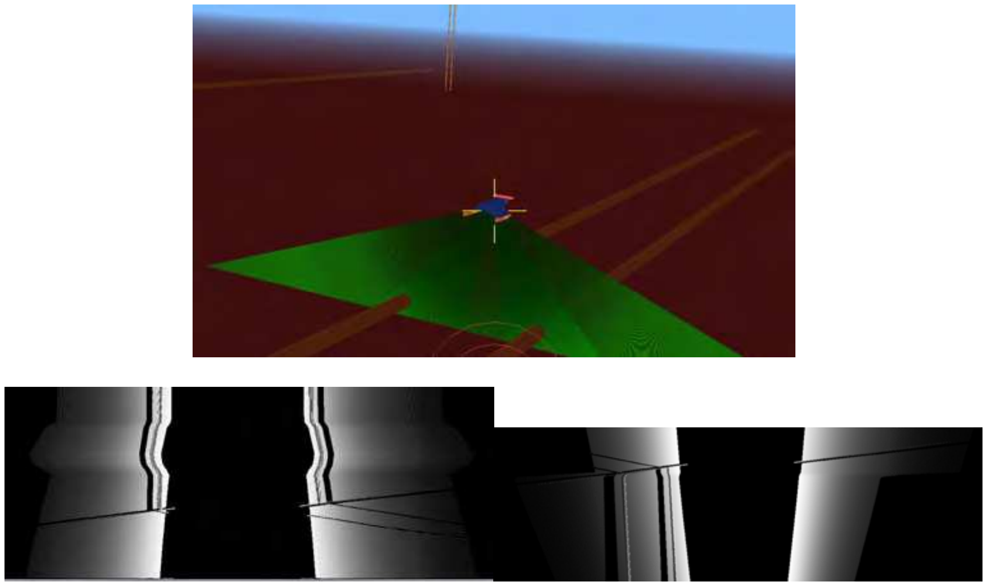


Fig. 9. Left & Right Sidescan Sonar simulation using ARF.

4.4 Nessie III

Nessie III (see figure 10) is the Ocean Systems Lab's entry to the 2008 Student Autonomous Underwater Vehicle Competition Europe (SAUC-E). The tasks the AUV had to execute included:

- Searching for the ground targets (tyres, drop-target, cones)
- Searching for the mid-water targets (orange and green balls).
- Touching orange ball
- Dropping weights on bottom target
- Surfacing in surface zone designated on the tangent between two tyres or two cones.

Specific behaviour routines are needed for lining up with the drop target for dropping the drop weights and for lining up with the orange ball ready for touching. This is straightforward if object positions are accurate and vehicle navigation doesn't drift. However, object positions are subject to inaccuracies due to sensor accuracy, false detections and navigational drift. Thus once an object has been detected, take for example the orange ball, the mission planner will change its task to touching the orange ball so that the positional drift is minimal. Rather than searching for all objects first and then going back for a closer inspection later. Positions of detected objects are stored by a world model on Nessie, so that once higher priority tasks are completed, the AUV goes to roughly where it thinks that object should be and starts a new search pattern to relocate it. Figure 10 shows the AUV homing in on the orange ball to touch it. ARF is used to visualise what the robot thinks is happening. ARF plots the detection of the orange ball and moves the virtual AUV accordingly.

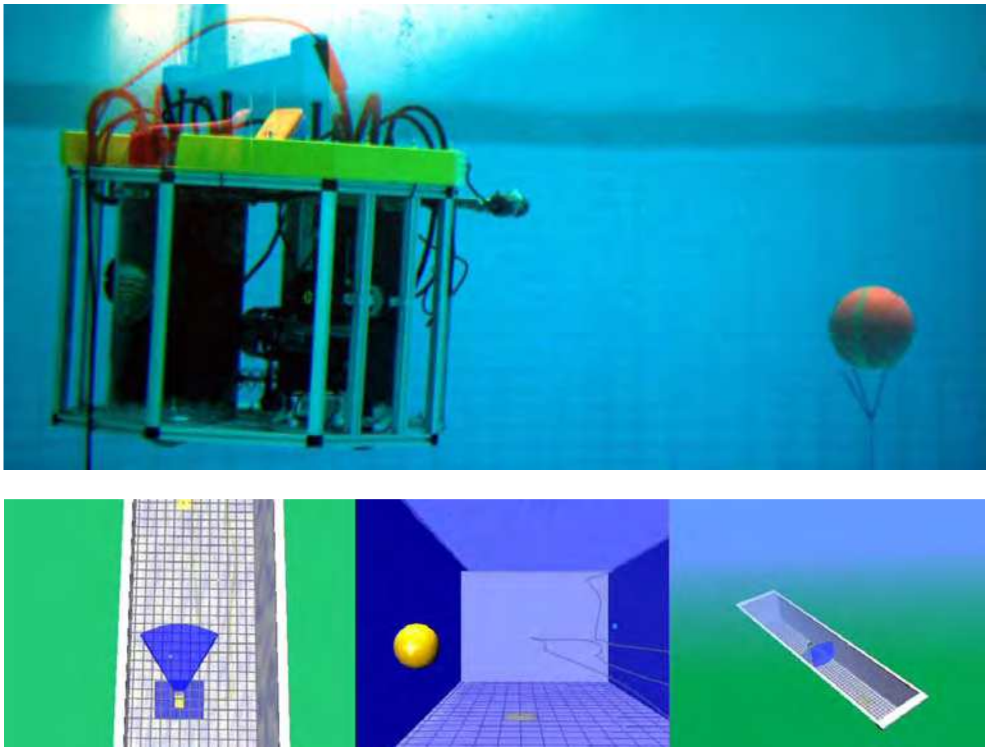


Fig. 10. Nessie tracking the orange ball for real and in simulation.

The Nessie III platform relies on visual tracking of the ground in order to estimate its movement and absolute position. In addition Nessie III is equipped with a Doppler Velocity Logger (DVL) which, combined with a compass, provides accurate navigation when visual systems cannot be used. Visual systems rely on the water being clear and well illuminated, but this is rarely the case in real environments. In addition to visualisation purposes ARF was used to test the behaviour of the autopilot, mission planner and Simultaneous Localisation and Mapping (SLAM) systems. A hydrodynamic model was used to simulate the movement of Nessie and a simple object detection sensor was programmed with the same parameters as the real vision systems of Nessie III. This enabled the real platform to execute its mission using the real autopilot and, simultaneously, test the mission planner to see if it made the right decisions and did the correct search behaviours under different circumstances. The output of the detection sensors was used to help position the AUV when the same object was re-detected using SLAM. However, if the vision system which provided the ground tracking on Nessie III failed the only source of navigation would be dead reckoning. This alone was not accurate enough to collide with the targets in the competition. Therefore, for the case of the orange ball especially, visual servoing could be used home in on the ball and bring the AUV into contact with it. Tank testing time was limited so ARF was used to tune the visual servoing method of the autopilot. The bottom left 2 images in figure 10 show the field of view of the forward looking cameras and the orange ball which is being tracked.

5. Performance testing

Use-cases demonstrate ARF's many different uses, and how extendable it is due to the JavaBean component based architecture. The performance increase of using ARF to create testing scenarios whilst taking into account its performance optimisations, such as guided construction, rapid configuration and large code reuse, is unknown. This can be quantified by comparing it to the standard approach of programming a test environment by hand. Since ARF uses JavaBean classes, it is relatively straightforward to compare the speed at which scenarios can be created using ARF against scenarios created using the same JavaBean classes but programmed by hand.

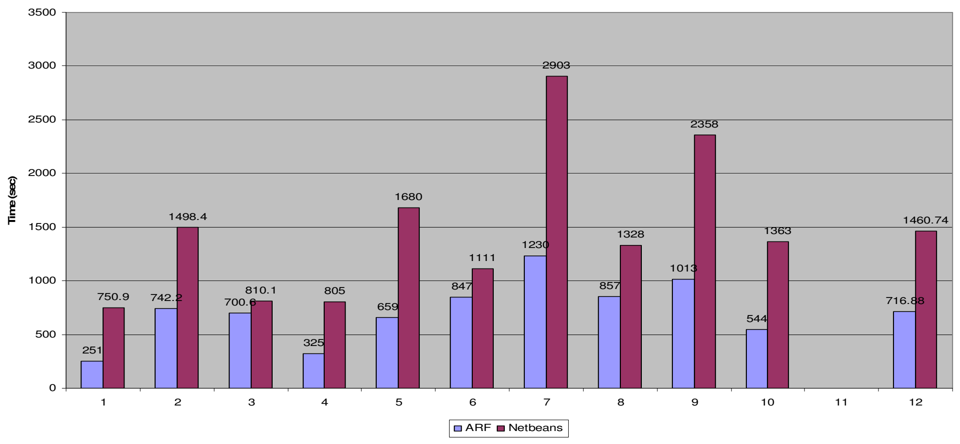


Fig. 11. Scenario creation times: ARF vs. Netbeans

For a fair test, the exact same JavaBean classes which are used in ARF will be used by the programmers. The programmers will use the NetBeans IDE as this provides help when

programming by showing the user relevant documentation of methods for classes. Also a full API specification will be available to programmers. The main task is to create a small virtual environment and connect the components together to simulate a basic AUV. The exact same components should be connected via programming using NetBeans. Measures of performance are gained from statistical feedback from the test participants as well as the tasks being timed. To make the test more useful, all test participants will be novices in using ARF and have only seen a brief video demonstration of how to use it, but are experts in programming. Other performance indicators, such as the amount of help and documentation required, are also logged. Figure 11 clearly shows the increase in performance of ARF compared with Netbeans. Number 12 on the graph shows the average values for all test participants which equates to on average ARF being over twice as fast as Netbeans. Generally, ARF would be expected to be even faster, however, because users were unfamiliar with ARF they took longer to do certain tasks. The times given above were to create a simple scenario with an AUV. Due to ARF's SuperComponent creation tools, the users were able to save their AUV configuration and then add a clone to the same environment and then adjust the properties so it was slightly different. The average time taken by the users to do this in ARF was *47 seconds*. Furthermore, the average time taken to make an alteration and save as a new configuration using ARF was only *37 seconds* compared with *77 seconds* in NetBeans. The ability to reuse large sections of code easily via the use of SuperComponents supports the "create once, use many times" ideal.
All these small increases in performance lead to greater increases in performance over a longer period of sustained use of ARF. The availability of such testing facilities helps the platform being developed become more mature within a shorter time frame. Different vehicle configurations can be easily created allowing for testing of potentially new vehicle configurations to see if they are viable and thus helping to develop the optimal solution of vehicle for a specific task.

7. Conclusions and further work

The testing carried out by the ARF Performance Tests (APT) highlight the features of ARF which are difficult to prove with only use cases. The most important points to draw from the results discussion is that ARF provides an extendable architecture which is generic enough to provide the capabilities of all the different usages. In essence ARF is a one size fits all architecture which provides generality on different granular levels providing flexibility for both programmers and high end users. The second most important feature that the APTs highlighted is the considerable performance improvement over conventional methods of scenario creation. Furthermore, the trade off of using ARF is minimal due to being built on existing JavaBeans technology so the programmer has little work to do to interface with ARF. The re-configurability of ARF is fast due to the use of simple project creation, SuperComponents and Guided Construction, which all help to increase programmer efficiency.
The analysis of the performance of ARF provides the fundamental evidence for why ARF is so powerful for higher level applications, such as HIL testing scenarios. The many small time savings which ARF provides propagate to massive time savings in more complex

projects. Scenarios no longer need to be programmed from scratch, they can simply be extended, have new components bolted on and merged with functionality from completely different projects through the use of SuperComponents. As ARFs Java3DBean and JavaBean component base grows, the more quickly scenarios can be created with ever more useful SuperComponents. The run away effect of having a large ever increasing component base means that before long design of new components will hardly ever be necessary. Further work is being carried out in increasing ARF's component base and expanding the usages of ARF into domains and fields other than sub-sea.

8. References

Choi, S.K.; Yuh, J. (2001).A virtual collaborative world simulator for underwater robots using multidimensional, synthetic environment. *In Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on Volume 1, 2001.* pp.926-931

Healey, A. J.; Pascoal, A. M.; Pereira, F. L. (1995) Autonomous underwater vehicles: An application of intelligent control technology: *Proceedings of the American Control Conference, Seattle, Washinton June 21-23, 1995,* pp. 2943-2949

Hirakawa, M.; Ichikawa, T. (1992). Advances in visual programming, *Proceedings of the Second International Conference on 15-18 June 1992 ICSI '92,* pp. 538-543

Lane, D.M.; Falconer, G.J.; Randall, G.; Edwards, I. (2001) Interoperability and Synchronisation of Distributed Hardware-in-the-Loop Simulation for Underwater Robot Development: Issues and Experiments. *Proceedings of the 2001 IEEE International Conference on Robotics and Automation, Seoul, Korea, May 21st – 26th*

Ocean Systems Lab. (2008). *OceanSHELL: An embedded library for Distributed Applications and Communications.* Heriot-Watt University

Patrón, P.; Smith, B.; Pailhas, Y.; Capus, C. & Evans, J. (2005). Strategies and Sensors Technologies for UUV Collision, Obstacle Avoidance and Escape: *7th Unmanned Underwater Vehicle Showcase September 2005*

Patrón, P.; Evans, J.; Brydon, J. & Jamieson, J. (2006). AUTOTRACKER: Autonomous pipeline inspection: Sea Trials 2005: *World Maritime Technology Conference - Advances in Technology for Underwater Vehicles, March 2006*

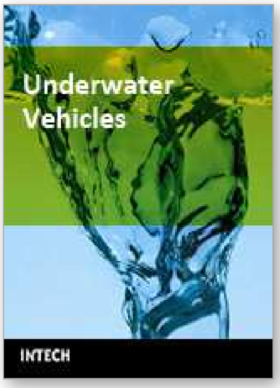
Pêtrès, C.; Pailhas, Y.; Patrón, P.; Petillot, Y.; Evans, J. &Lane, D.M.; (2007) Path Planning for Autonomous Underwater Vehicles: *proceedings of IEEE Transactions on Robotics, April 2007*

Ridao, P.; Batlle, E.; Ribas, D.; Carreras, M. (2004). NEPTUNE: A HIL Simulator for Multiple UUVs: *In proceedings of OCEANS '04. MTS/IEEE TECHNO-OCEAN '04 Volume 1, 9-12 Nov. 2004 pp.524-531*

Sotzing, C.C.; Evans, J. & Lane, D.M. (2007). A Multi-Agent Architecture to Increase Coordination Efficiency in Multi-AUV Operations: *In proceedings IEEE Oceans 2007, Aberdeen;*

Sun Microsystems, JavaBeans, *http://java.sun.com/javase/technologies/desktop/javabeans*

Zyda, M.J.; McGhee, R.B.; Kwak, S.; Nordman, D.B.; Rogers, R.C.; Marco, D. (1990). Three-dimensional visualization of mission planning and control for the NPS autonomous underwater vehicle: *Oceanic Engineering, IEEE Journal Volume 15, Issue 3,* pp.217–221



ISBN 978-953-7619-49-7
Hard cover, 582 pages
Publisher InTech
Published online 01, January, 2009
Published in print edition January, 2009