Simulation of Underwater Robots using MS Robot Studio©

John Prevost, *Student Member* IEEE, Matthew A. Joordens, *Member IEEE* and Mo Jamshidi, *Fellow* IEEE
Autonomous Control Engineering (ACE) Center and ECE Department
The University of Texas
San Antonio, TX, USA
matthew.joordens@utsa.edu

***Abstract*—One stage in designing the control for underwater robot swarms is to confirm the control algorithms via simulation. To perform the simulation Microsoft's Robotic Studio© was chosen. The problem with this simulator and others like it is that it is set up for land-based robots only. This paper explores one possible way to get around this limitation. This solution cannot only work for underwater vehicles but aerial vehicles as well.**

***Index Terms*— Simulation software, Robots, Underwater vehicles**

I. Introduction

The University of Texas at San Antonio's Autonomous Control Engineering (ACE) lab has the objective to build and deploy Land, Air and Sea autonomous robotic vehicles that can independently operate toward a common pre-programmed objective.

To facilitate this objective, three separate projects have been undertaken: Land, Air and Sea. Working in all these areas it is important that this System of Systems (SOS) of robots be able to be simulated in order to see how they integrate together.[1] This is especially true as it is not always possible to provide a controlled environment that has land, air and sea for initial SOS experiments.

The Sea project will build and deploy underwater robots that can independently operate as part of a functional team by actively probing the environment to understand parameters such as relative position, global position (GPS), depth, speed, and proximity to other underwater entities such as other robots and/or the sea floor.

In order to efficiently program and test these robots, it is desired that a realistic simulation environment be created that can be used as a "virtual" underwater staging area. The programmed robots will then be able to be tested in a variety of scenarios, without having to incur the time and expense required to test the robots in an actual physical environment. This work was performed to simulate the underwater robots as described in Joordens, et al.[2]

Whilst this paper is looking primarily at the simulation of underwater robots, the ultimate goal is to perform simulation of land, air and sea robotic swarms all in the one simulator.

II. Simulator Choice

There are various simulators available. One such simulator is ROBOSIM. This simulator is windows based, something that was desired as the available computers, including the one's on the actual robots are windows based. ROBOSIM, however was designed for articulated join robots, whereas multi-robot swarms was required. Also LISP is the main programming language which none of the researches had used. The limited Graphical User Interfaces (GUI) was another factor that reduced the desirability of this package.[3]

AutoLisp can be used with AUTOCAD to model and simulate robots. The inclusion of AUTOCAD allows for simple modeling of individual robots but it lacks a simple way to create an environment for the robots to operate in.[4] This factor and the unfamiliarity with AutoLisp made this approach less than desirable.

JAVA and JAVA3D was another possibility. This system used the JAVA programming language[5] and the JAVA API's for 3D. Whilst this has specific functions to model 3D objects and including functions like collision detection it does not include a full physics engine.[6]

The MOBILE software package is good at simulating robots that operate in parallel such as the PARTNER robots.[7] Unfortunately it was not as user friendly as was liked.

A Different approach was to build our own simulator as was done by the department of Computer Science and Engineering, Washington University in St. Louis. Ogre3D can provide a graphical environment and Ageia's PhysX can be used as the physics engine.[8] Ageia's PhysX physics engine is an excellent product that can work in any environment that is chosen. Another engine is the Open Dynamic Engine Library (ODEL).[9] The Technical University of Cluj-Napoca, Romania also built their own using C++ and OpenGL.[10] This build your own approach has one particular advantage. It can be used for underwater and aerial simulation as well as land base simulation. All the other simulation systems are

This work was supported in part by University of Texas, San Antonio, Autonomous Control Engineering and by Deakin University.

J.J. Prevost is with Injury Sciences, LLC San Antonio, Texas, USA (phone: (210) 691-0674; fax: (210) 699-1880; e-mail: jeffp@injsci.com)

M. A. Joordens is with the School of Engineering and Technology, Deakin University, Geelong, 3217, Australia (phone: +61-3-52272824; fax: +61-3-52272167; e-mail: matthew.joordens@deakin.edu.au).

978-1-4244-2173-2/08/$25.00 ©2008 IEEE

designed for land based simulation only. Unfortunately, the build your own approach will involve a large amount of development time that could not be spared in the SOS research timeframe.

The National Institute of Multimedia Education, Osaka University has developed an online robot simulator call Robot Studio.[11] This was a decent web based simulator but didn't have the flexibility required as it was designed primarily for education.

A possible framework for simulation is the Ubiquitous Robot Simulation Framework (URSF). This simulated ubiquitous robots that operate in a global environment.[12] This framework meets the requirements of swarm robotics but was just not user friendly enough.

A combination of ADAMS and MATLAB simulation allows the familiar use of the MATLAB interface.[13] This is another system that is land based only but could possibly be modified to suit.

USARSim is a system that builds on the Unreal Game Engine for the graphics and the physics engine. An added feature of this simulation system is that the developers are looking at simulating a submarine.[14] The submarine is a single propeller design with diving planes whereas the robots to be simulated are of a multi-thruster design. The learning curve was fairly steep.

Microsoft's (MS) Robotic Studio (not to be confused with Osaka University's Robot Studio) is a windows based simulator that uses the Ageia's PhysX physics engine. It is programmed with MS Visual Studio C#. It has many of the advantages of the self developed simulator but the disadvantage of being a land based simulator.

III. Programming/Simulation Environment

It was determined that the Microsoft Robotics Studio (MSRS) SDK add-in to Visual Studio would be the best programming/hosting environment to create and simulate the robots.

Microsoft created the MSRS to allow researchers, developers and industry to have a robust development platform for creating and simulating robots using industry-standards such as the C# programming language, XML and web-services. This environment allows the robot designer to be able to design the robots virtually and to simulate them in any environment before the robot is actually constructed. Once constructed, the code used in the simulator can be ported directly to the robot that will then behave as per the simulator. If any new situations appear, the designer can simulate this and perform all testing in the simulator before placing the robot in that situation.

At the core of the MSRS is the Decentralized Software Service (DSS). The DSS allows for a "service-model" approach to the software development. This means that the various parts required to control and run the robots can all be created modularly. The re-use of these code modules expedites the time required to deliver projects. Once created, the modules are registered into the DSSHost program as services that then can be subscribed to by any entity registered with the DSSHost. This allows for easy synchronization and coordination of the robots in real-space or virtual-space.

The entities that represent the individual functional components of the robots are created as modules and registered with the DSSHost. Examples of these modules are the Laser Range Finder, the Sonar Range Finder, the wheel motor drives, articulated joints, the thruster engines and GPS location devices. These modules communicate by utilizing the Concurrency and Coordination Runtime (CCR) using asynchronous messaging. For example, when the Laser Range Finder records an object in the path of the laser, it sends a message to the DSSHost through the CCR. Any entity that is listening to the Laser Range Finder Service gets notified of the message and can respond as appropriate. This asynchronous messaging paradigm allows for very high concurrency of events to be processed and should allow for a multiple robots to be simultaneously engaged.

At the heart of the developers tool-set is the Visual Programming Language (VPL). The VPL allows for a LabView like graphical modeling approach to create the entity behaviors that belong to each entity. The developer uses a palette of objects and behaviors and "drags and drops" them onto the VPL design surface (Fig. 1). By connecting the visual objects together and setting the exposed properties correctly, the developer can create a variety of distinct entity behaviors. These behaviors can then be set to trigger upon notification of the appropriate message from the DSSHost.

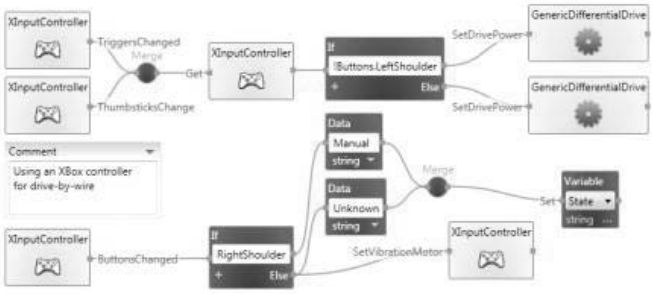

Fig. 1 Microsoft Visual Programming Language

The simulation environment is enabled by the AGEIA PhysX engine. This is a real world physics simulation engine licensed from Ageia Technology. The physics engine allows for accurate modeling and response of the robots to take place in the virtual environments. The robots can be initialized in a virtual world and move and respond to external stimuli just as if they would if they were deployed in a real world environment.

In order to create a simulation that can emulate an underwater environment, several parameters need to be explored.

The first thing to understand is how to create a bounding area that will contain water (our pool). Next, there needs to be a boundary restriction that will not allow the water entities to go out of the area defined as the pool (from the top). Once this is established, exploration of the available environmental physical properties must be explored so the robots can be

made neutral, or positively, buoyant. Another property of water that differs from air is viscosity. An object in water does not obey the same rebounding and restitution properties is it would if on land or in the air. Water has a damping effect that acts a friction opposing motion in the direction of force. Finally, it must be determined how to establish motion itself on the underwater entity.

Along with establishing the correct set of mechanics for an underwater environment, the submarine itself must be modeled so an accurate visual representation of the sub can be used in the simulation.

IV. Simulation Findings

To establish a ground plane that can be used for water, the HeightFieldEntity class was used. This class allows for a ground to be established by defining a matrix of heights, each separated by a defined column area.

The first approach taken was to establish two HeightFields. The upper HeightField was created at 3 meters, the lower created at zero meters. The EnableContactNotification property of the upper HeightField was then set to false. The underwater entities were then set to initialize at the zero height level. It was thought that this approach would allow for establishing the properties of water by setting the upper HeightFieldEntity properties appropriately. This did not prove to be a practical methodology. What happened at runtime was that the AGEIS PhysX Engine determined that "invalid" contact was being made between the robots and the ground-plane (the upper HeightField). Right after the robots appeared in the simulator, they were "bounced" hundreds of meters in the sky only to land (with a thud) way off in the distance. Although comical, it was far from being practical.

```
TerrainEntity ground = new
TerrainEntity(
      //"terrain.bmp",
      "pool2.bmp", // great pool
bitmap
       "terrain_tex.jpg",
        new
MaterialProperties("ground",
         2.0f, // restitution
         0.5f, // dynamic friction
          0.5f) // static friction
       );
```

Listing 1: TerrainEntity used for pool area

The next approach taken was to create a matrix of HeightFieldEntities with dimensions 60 X 60. Each cell was defined as being 1 meter apart from the adjacent cell. All the cells at the exterior of the matrix had a height set to 3 meters; the inner cells had their height set to zero meters. All the underwater entities would then be initialized at runtime to be inside the perimeter HeightFieldEntities at the zero height level. The robots appeared as they should at runtime, but the upper HeightField was not visible. When an upper translation boundary was created and initialized on top of the HeightField, the boundary simply fell through. The physical border properties of the HeightField simply did not support the loading of a new entity on top of it. This was deemed not to be sufficient.

Finally, it was discovered that a terrain mesh could be created that allowed for a variety of heights; all based on the color values in a bitmap image. A bitmap was created as n pixels by m pixels. This corresponds to a terrain n meters x m meters in the simulation environment. Next, the bitmap receives color values corresponding to the desired relative heights of the terrain. In our case, a square of white was created in the center of a solid black image. The white area represented our pool; the black area represented the ground level. On runtime, a nice depression appeared in a level terrain. The robots were then initialized inside the pool (listing 1) . The upper boundary was then created and initialized on the ground level, effectively creating a vertical translation boundary for our robots.

The next step taken was to create an actual object that would represent our underwater vehicle. The freeware program, Wings3D was used to create a 3-dimensional rendering of the yellow submarine in the ACE lab. The .obj file was exported from Wings3D and saved into the media directory of MSRS. In code, a rectangle entity was created with a visual mesh defined using the submarine .obj file. At runtime, there appeared a white version of the 3D model of the sub (Fig. 2). The problem now was that right after runtime; the sub disappeared through the bottom of the pool. It was soon discovered that there needed to be synchronization between the underlying physical entity and the visual representation. Once this was accomplished, the sub stayed on the pool bottom without falling through.

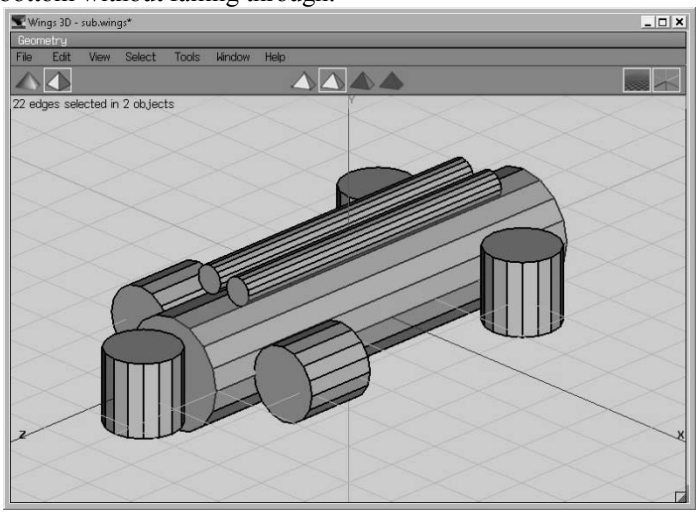

Fig. 2 3D model of submarine.

In water, it was desired to have the sub maintain positive buoyancy, or at least be neutral. After much time and effort, a public property named IgnoreGravity was discovered that seemed to work. Setting this property to true allowed the sub to be positioned midway between the floor and the upper

boundary. When the simulation was run, the sub stayed where it was initialized. Gravity had no effect on the sub, effectively acting as if the sub was maintaining neutral buoyancy.

The sub was then given an initial velocity by creating a vector of the desired meters per second as a 3D vector. When the simulation was run, the sub slowly moved along the appropriate path at the pre-set velocity.

The only thing remaining to establish a true underwater simulation was to somehow change the properties of the medium the sub was traveling through to emulate the viscosity present in water. There was no entity representing "air", but since the effect could be modeled as friction along a vector opposing the motion of the vehicle some form of damping property may be good enough. Two such properties were discovered, LinearDamping and AngularDamping. These two properties are accessible at runtime by putting the simulation in "edit" mode and manually setting the values. Unfortunately, there is seemingly no way of accessing the proper object layer at design time, making it impossible to initialize the environment with these values. When the properties are set at runtime, the sub reacts to being bumped in the predicted manner. It moves away from the applied force, but quickly loses momentum and comes to a stop.

The code used to create a single sub was now essentially complete, that is one sub can be simulated. In order for the swarm concept to be implemented, the code needed to be modified to allow for any number of sub objects to be instantiated in the simulator. This was accomplished by factoring the instance name, initial position vector and initial velocity vector. This allowed the method to be called with these as parameters, thereby allowing for it to be called once for each instance of a sub desired. Fig. 3 shows two subs running in the simulated underwater environment.

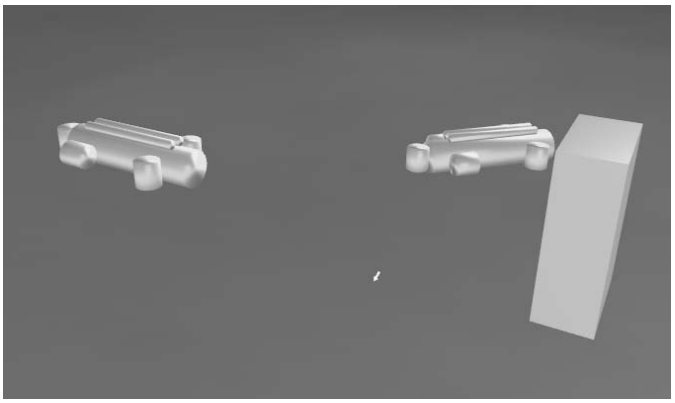

Fig. 3 Two submarines in the simulator.

V. Conclusion

The MSRS is currently designed to effectively model and simulate land-based robots. Through effort, however, one can create an underwater simulation that can be used to test the programming of robots without needing to do a real deployment. Some of this effort includes: needing to create a "top" so the robots are maintained in the proper water area and requiring the developer to set the responses to the environment on each deployed entity, such as the IgnoreGravity property. Even with this effort, there are limitations on what can be done. Among these are: seemingly no design-time way to set the damping properties of an entity and no way to create a naturally positively buoyant environment for the robots.

The results of this project were eventually positive. The lab sub was modeled and placed in an underwater environment where a simulation can now be performed. The next step would be to create the thruster objects that could be registered with the DSSHost so they could be controlled via VPL. Then the individual sensor objects would be created and registered as well. Once this has been accomplished, the sub will not only be able to be programmed for real world activity, but the sub programming will be able to be tested in a cost effective and expedient manner.

Finally, the system developed here can be easily ported to adapt the Robotics Studio to simulate aerial robots, as the aerial environment is also a 3 dimensional environment with the main medium being air instead of water.

References

[1] E. Azarnasab and X. Hu, "An Integrated Multi-Robot Test Bed to Support Incremental Simulation-Based Design," *System of Systems Engineering,* pp. 1-7, 2007.

[2] M. Joordens, "Design of a low cost Underwater Robotic Research Platform," in *aper submitted to IEEE SoSE Confernec, Monterey, CA, USA*, Monterey, CA, USA, 2008, p. 6.

[3] Z. Bingul, P. Koseeyaporn, and G. E. Cook, "Windows-based robot simulation tools," *Control, Automation, Robotics and Vision,* vol. 2, pp. 1037 - 1041, 2002.

[4] Y. Yan, C. Xing, L. Chang-hua, and K. Bo-seon, "A Robot Simulation System Basing on AutoLisp," *Industrial Electronics and Applications,* pp. 2154 - 2156, 2007.

[5] X. Pan, Y. Chen, and L. Yan, "Research of Multi-robot Cooperation Simulation and Monitoring System Based on Java," *Intelligent Control and Automation,* vol. 2, pp. 9494 - 9498, 2006.

[6] D. Zhijiang, Y. Donghai, S. Lining, and F. Lixin, "Virtual Robot Simulation and Monitoring System Based on Java3D," *Robotics and Biomimetics,* pp. 313 - 317, 2004.

[7] C. Brisan, M. Hiller, and I. Birou, "Virtual Models of Special Class of PARTNER Robots," *Automation, Quality and Testing, Robotics,* vol. 2, pp. 266 - 271, 2006.

[8] J. Faust, C. Simon, and W. D. Smart, "A Video Game-Based Mobile Robot Simulation Environment," *Intelligent Robots and Systems,* pp. 3749 - 3754, 2006.

[9] R. Aubin, P. Blazevic, B. Clement, and J.-P. Guyvarch, "Simulation and Design of a Snake-Like Robot Based on a Bio-Inspired Mechanism," *Biomedical Robotics and Biomechatronics,* pp. 220 - 225 2006.

[10] C. Marcu, G. Lazea, and R. Robotin, "An OpenGL Application for Industrial Robots Simulation," *Automation, Quality and Testing, Robotics,* vol. 2, pp. 254-259, 2006.

[11] J. Nakahara, A. Imai, M. Taguchi, A. Itou, Y. Yamauchi, and T. Sunaga, "The development of web-based learning community system to facilitate learner's creating the autonomous robots," *Computers in Education,* vol. 1, pp. 409 - 410, 2002.

[12] M. Jang, J. Kim, M. Lee, and J.-C. Sohn, "Ubiquitous robot simulation framework and its applications," *Intelligent Robots and Systems,* pp. 4077 - 4082, 2005.

[13] Y. Yi, F. Mengyin, S. Changsheng, W. Meiling, and Z. Cheng, "Control Law Design of Mobile Robot Trajectory Tracking and Development of Simulation Platform," in *Control Conference*, 2007, pp. 198 - 202.

[14] S. Carpin, M. Lewis, J. Wang, S. Balakirsky, and C. Scrapper, "USARSim: a robot simulator for research and education," *Robotics and Automation,* pp. 1400 - 1405, 2007.

Matthew A. Joordens (Member -IEEE, Fellow - The Institution of Engineers Australia) earned his Bachelor of Engineering (electronic) degree at Ballarat University in 1988 and a Masters of Engineering (by research) in Virtual Reality at Deakin University in 1996.

He began his career with Industrial Control Technology designing control systems to automate various different industrial processes. For 5 years he designed microprocessor based control systems for companies such as Ford, Pilkington Glass, Webtek and Blue Circle Southern Cement. He then moved to Deakin University and wrote their first electronics units. Using his industrial experience he designed one of the first Australian Engineering degrees in Mechatronics that still runs at Deakin as Mechatronics and Robotics. He currently lectures units in Digital electronics, Microcontrollers, Robotics and Artificial Intelligence after 15 years at Deakin. He is currently researching underwater swarm robotics in the USA.

Mr. Joordens is a Fellow of the Institution of Engineers, Australia and an IEEE member.

Mo M. Jamshidi (Fellow-IEEE, ASME, AAAS, NYAS, TWAS, Associate Fellow AIAA) received his PhD Degree in Electrical Engineering from the University of Illinois at Urbana-Champaign in February 1971. He holds Honorary Doctorate Degrees from Azerbaijan National University, Baku, Azerbaijan (1999), the University of Waterloo, Canada (2004)and Technical University of Crete, Greece, (2004). Currently, he is the Lutcher Brown Endowed Chaired Professor at the University of Texas at the San Antonio Campus, San Antonio, Texas, USA. He was the founding Director of the Center for Autonomous Control Engineering (ACE) at the University of New Mexico (UNM), and moved to the University of Texas, San Antonio in early 2006. He has been the Director of the ICSoS – International Consortium on System of Systems Engineering (i*csos.org*) – since 2006. Mo is an Adjunct Professor of Engineering at Deakin University, Australia. He is also Regents Professor Emeritus of ECE at UNM. In 1999, he was a NATO Distinguished Professor in Portugal of Intelligent Systems and Control. Mo has written over 560 technical publications including 62 books and edited volumes. Six of his books have been translated into at least one foreign language other than English. He is the Founding Editor, Co-founding Editor or Editor-in-Chief of 5 journals. Mo is Editor-in-Chief of the new *IEEE Systems Journal* (to be inaugurated in 2007) and founding Editor-in-Chief of the *IEEE Control Systems Magazine*. Mo is a Member of Senior Member of the Russian Academy of Nonlinear Sciences, Hungarian Academy of Engineering and an associate fellow of the AIAA. He is a recipient of the IEEE Centennial Medal and IEEE CSS Distinguished Member Award. Mo is currently on the Board of Governors of the IEEE Society on Systems, Man and Cybernetics and the IEEE Systems Council. In 2005, he was awarded the IEEE SMC Society's Norbert Weiner Research Achievement Award and in 2006 the IEEE SMC Distinguished Contribution Award. In 2006 he was awarded a "Distinguished Alumni in Engineering" at the Oregon State University (one of his Alama Matters), Corvallis, Oregon, USA