Creating Virtual World Environments for Ocean Vehicles

Ryan Capozzi, Amanda Costa, Ian Friedrichs
Unmanned Systems, a business group within HII's Mission Technologies division
Pocasset, MA
Ryan.Capozzi@hii-tsd.com, Amanda.Costa@hii-tsd.com, Ian.Friedrichs@hii-tsd.com

ABSTRACT

Virtual test environments are critical for supporting vehicles in all domains throughout their lifecycle, with simulation frequently being called upon for tasks ranging from concept feasibility studies and new feature development to requirements validation, predictive maintenance, and customer support. High fidelity simulation of ocean environments allows vehicles to execute missions in unforeseen environments or known operating conditions that are challenging to reproduce during in-water testing. The benefit of such virtual environments is directly related to not only the simulation's accuracy, but also the speed at which a simulation can be configured. The addition of game engines such as Unity and Unreal Engine allow high fidelity simulations to be configured rapidly while using high detail models and providing capability critical for simulation of sensors and world models. While significant capacity for simulating land and air vehicles exists in these platforms, the options become far more limited when simulating vehicles at sea. This paper will discuss methods for using Unreal Engine 5 to develop high-fidelity virtual world models for marine vehicle simulation, with a particular focus on simulations for HII's REMUS unmanned underwater vehicles (UUV) product line. HII will present approaches and workflows for creating test environments capable of verification and validation of oceanic vehicles including real-world and synthetic bathymetry import, vehicle visualization, stationary and dynamic obstacles, example sensor development, and built-in subprocesses to handle flow field effects (dynamic temperature and salinity profiles, ocean turbulence, ocean current, and irregular Sea State wave effects), all while leveraging ROS2 with Unreal Engine.

ABOUT THE AUTHORS

Ryan Capozzi is a Modeling & Simulation engineer at Unmanned Systems, a business group within HII's Mission Technologies division. He holds a MS in Aerospace Engineering from Florida Institute of Technology and a BS in Aerospace Engineering from Worcester Polytechnic Institute. Ryan has designed and developed software for REMUS platforms and is currently a lead developer of digital twins for such systems.

Amanda Costa is the Head of Modeling and Simulation at Unmanned Systems, a business group within HII's Mission Technologies division. Amanda holds her MS Degree in Ocean Engineering from the University of Rhode Island and her BS in Physics and Mathematics from Merrimack College. She has 10 years of professional experience developing hydrodynamic solvers for maritime vehicles from large Naval ships to small-class underwater vehicles and integrating them into early ship design, manufacturing, and customer service processes.

Ian Friedrichs is a Modeling & Simulation engineer at Unmanned Systems, a business group within HII's Mission Technologies division. He graduated from Stevens Institute of Technology with a BS in Physics and is pursuing a MS in Data Science at Eastern University. Ian's focus includes the development of simulation tools for unmanned maritime vehicles, the use of Computational Fluid Dynamics (CFD) to support their design, and the integration of the two platforms.

Creating Virtual World Environments for Ocean Vehicles

Ryan Capozzi, Amanda Costa, Ian Friedrichs
Unmanned Systems, a business group within HII's Mission Technologies division
Pocasset, MA
Ryan.Capozzi@hii-tsd.com, Amanda.Costa@hii-tsd.com, Ian.Friedrichs@hii-tsd.com

INTRODUCTION

Virtual testing has become essential for vehicles throughout their lifecycle. The proper use of virtual testing improves design decisions, informs maintenance, and performs validation across all domains. While there is no true substitute for sea testing, underwater vehicles have additional challenges that are particularly well suited for virtual testing. The nature of subsurface vehicles may limit the ability to perform visual verification of desired operation during a mission, and local bathymetry may make certain vehicle missions impossible to test locally. Many of those missions that cannot be conducted locally are critical to validate key performance parameters. Additionally, subsurface vehicles can be lost if safety critical functions fail, and while some shore testing can be done to exercise these capabilities, fault injection using a virtual environment provides a better analogue to failure during a mission. Operator training and vehicle usability is also positively impacted by virtual testbeds, as a desired mission can be simulated over imported real bathymetry. This allows both vehicle capability and mission design to be confirmed prior to launching an asset.

HII Unmanned Vehicle Maritime Simulation Environment

HII develops, maintains, and leverages an underwater simulation environment to support the design, development, integration, and ongoing test of unmanned vehicles. The simulation environment is built on the 3D virtual world platform Unreal Engine (Epic Games, 2024) depicted in Figure 1 below. The platform supports bathymetry imports from real-world sources, stationary and dynamic obstacles, and enables interactive sensor modeling. There is also support for importing fluid velocity fields output from external Computational Fluid Dynamics (CFD) software. These features are user configurable and adjustable to mimic various realistic environments. This flexibility enables execution of vehicle missions in unforeseen environments or known operating conditions that are hard to reproduce with in-water testing.

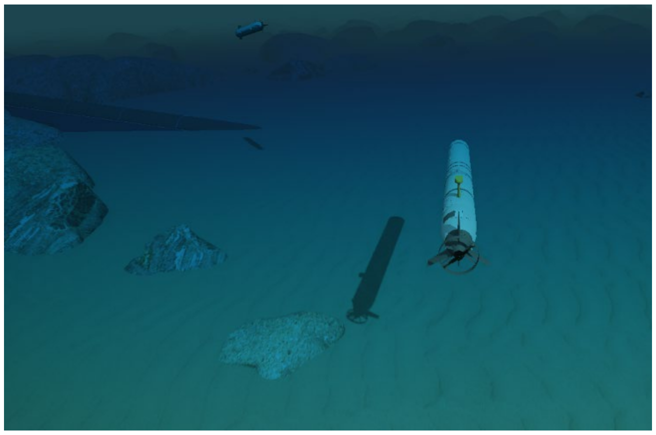
Figure 1: Simulated Small- and Medium-class UUVs operating in HII's Undersea Simulation Environment using actual bathymetry of Buzzards Bay, MA (Platform: Unreal Engine)

Paired with the virtual world, HII developed a modular simulation environment built on the Robot Operating System (ROS) 2 platform (Open Robotics, 2024), with a high-fidelity physics solver at its core. The modular architecture allows for a configurable simulated vehicle represented by the hydrodynamic hull signature and actuators from CFD and outfitted with simulated sensors. The sensors are mathematical representations with implemented specifications based on vendor data sheets and real-world data. The vehicle model interacts with the virtual world through its sensors and hydrodynamic properties as shown in Figure 2. The high-fidelity physics solver uses these properties to solve a set of non-linear equations integrated in time for the dynamic vehicle state.

The combination of virtual world and physics-based simulation environment represent a modular virtual testbed that enables efficient measurement of vehicle performance and complete analysis of alternatives regarding design or operating conditions. This provides an efficient analysis platform for vehicle design modification.

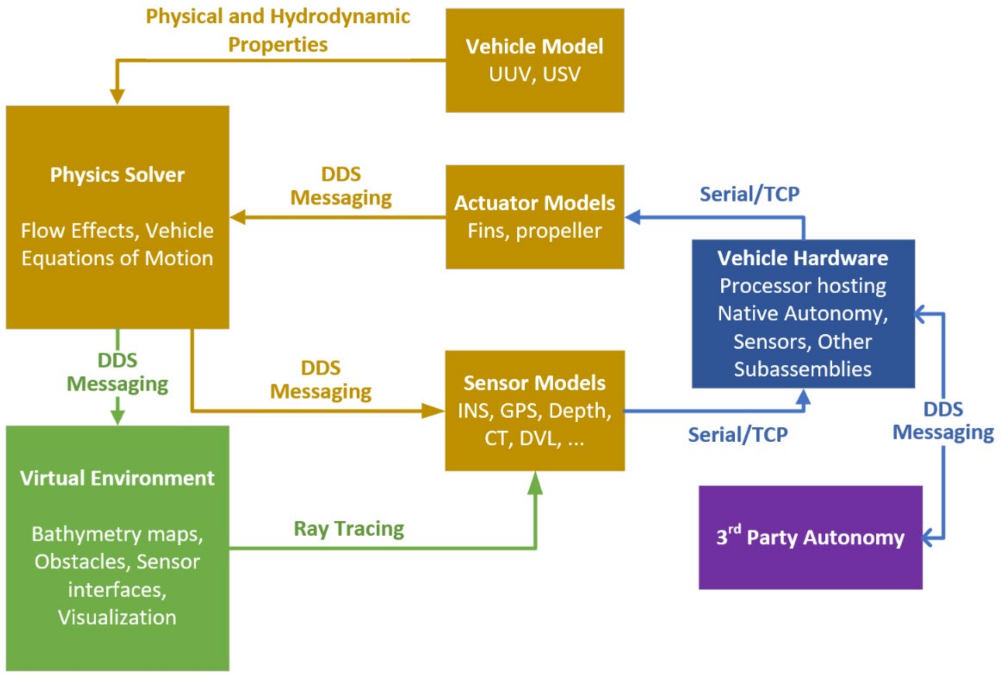
Physical and Hydrodynamic Properties
Vehicle Model
UUV, USV
Physics Solver
Flow Effects, Vehicle Equations of Motion
DDS Messaging
Actuator Models
Fins, propeller
Serial/TCP
Vehicle Hardware
Processor hosting Native Autonomy, Sensors, Other Subassemblies
DDS Messaging
DDS Messaging
Sensor Models
INS, GPS, Depth, CT, DVL, ...
Serial/TCP
DDS Messaging
Virtual Environment
Bathymetry maps, Obstacles, Sensor interfaces, Visualization
Ray Tracing
3rd Party Autonomy

Figure 2: Architecture of HII's unmanned vehicle maritime simulation environment.

The architecture diagram in Figure 2 details the high-level components of the simulation environment as well as the communication between components. The virtual environment shown in green is essential for providing environmental information to the sensor models. Environmental effects on the simulated vehicle are processed by the physics solver after passing through vehicle hardware and actuator models. The architecture presented allows for fully integrated, real-time simulation of the vehicle within the virtual environment. Each orange block in Figure 2 represents ROS2 node(s) where a node is created for each vehicle model, each actuator, and each sensor separately to maximize simulator modularity. Each vehicle receives its own physics solver node with communication handled as messages and services within ROS2.

Unreal Game Engine for Simulation

Game engines include many capabilities key to simulating vehicles of all types. The ease of importing models and creating test scenarios coupled with built-in collision detection and ray tracing make them powerful tools for simulation frameworks. Built in ray tracing greatly simplifies development for a vast number of sensors and allows them to interact with synthetic or real bathymetry maps. In addition, both land and air vehicles can often directly utilize the engine's physics model and should these no longer suffice, there are additional options for higher fidelity physics solvers for these platforms. The addition of a new water system in Unreal Engine 5 is an improvement over previous generations and can be useful for low-level surface vessel simulation. For underwater vehicles, it becomes necessary to couple an engine such as Unreal with a specialized physics solver. This paper will discuss the creation and simulation of sensors and actuators, the import of assets and vehicles into Unreal Engine, the maneuvering solver for such vehicle assets, and acquisition and import of real bathymetry maps for the creation of realistic test scenarios.

UNREAL VISUALIZATION PLATFORM

Unreal Engine is developed by Epic Games, Inc. and was first introduced in 1998. Since then, new features have been added and it has grown from an engine whose primary use was for developing first-person shooter games into a platform capable of importing vehicle CAD models, GIS landscape import, collision detection and import of 3rd party assets. A few common uses for Unreal Engine include virtual reality maintenance training, mixed reality learning applications, multi-car simulation environments, and flight simulators.

Generating and Importing Assets

Vehicle models are imported into Unreal using a two-step process. Some versions of SolidWorks are compatible with Unreal Engine's Datasmith plugin, which will likely be the easiest method for most vehicle models. It is useful to note that regardless of import method used, a stripped-down version of the desired vehicle should be used to reduce complexity of the resulting static mesh. Below, we outline a different workflow for model import currently used by HII's simulation environment such that Datasmith is not required.

The workflow for static mesh conversion from a CAD program such as SolidWorks is straight-forward even without additional plugins, though somewhat more tedious. First, export the vehicle as a STL or WRL file. Blender, an opensource 3D computer graphics tool, can be used for conversion. At the time of writing Blender does not support the latest WRL format. As a result, when exporting a WRL file from SolidWorks for conversion using Blender, the older VRML97 format must be used. Exporting the vehicle by color or desired visual texture is beneficial during this export step if accurate color is desired in the final product. Next, import all sections of the vehicle and add a different material for each section (naming the materials in this step will make it easier to add materials later). Finally, save the vehicle as a FBX file and import into Unreal Engine. This gives an accurate vehicle model that can be outfit with sensor models and used with collisions.

Creating Stationary and Dynamic Obstacles

Stationary obstacles are handled entirely within the engine itself. They are easily added to a simulation by using the built-in editor or by spawning obstacles of interest programmatically. Stationary obstacles cover a vast array of applications including rocks and shipwrecks, generation of some synthetic geometries for testing vehicle response, as well as addition of common targets, such as subsurface mines. Adding the programmatic ability to spawn such stationary obstacles requires minor effort and greatly increases the types of simulations that can be run. Unreal Engine provides both JSON and INI utilities built in to assist with file parsing.

Dynamic obstacles can be implemented in one of two ways. The primary method of moving dynamic obstacles is identical to the vehicle model, the obstacle state is updated by the external physics solver over ROS2. Lower fidelity dynamic obstacles not requiring such a solver can be created entirely within the Engine using a pawn and moved by setting the pawn's velocity.

Bathymetry Import

Bathymetry import is crucial for enabling virtual testing in key locations around the globe or troubleshooting unexpected vehicle behavior observed at sea. Real bathymetry data around the United States is provided at easy access by NOAA in XYZ format (NOAA, 2024). These files contain three fields of interest: latitude, longitude, and depth. While there are many common file formats for chart import, using alternate filetypes changes little except for file parsing. There are two major methods for importing bathymetry: raster methods and mesh methods. Both methods have support from open-source projects, with raster methods leveraging GDAL (Warmerdam & Rouault, 2024) and mesh methods having access to Open3D (Open3D, 2024).

Raster Methods

Importing into Unreal Engine using the raster method relies on the landscape plugin in Unreal Engine 5, however raster import for landscape creation is common across game engines and will be supported by a vast array of modern engines. The process for import using Unreal with GDAL is only a few steps:

1. Open the filetype of choice and convert to CSV file format.
2. Generate VRT file including CSV file and column names.
3. Use gdal_grid to convert CSV to TIFF image and return both maximum and minimum values.
4. Use gdal_translate to convert TIFF to PNG image.

From this point, it is possible to manually load the image into Unreal Engine using the editor and an open world map. Alternatively, this bathymetry data can be automatically imported by creating a new actor and taking advantage of Unreal Engine's existing ALandscape class.

Mesh Methods
Using the Mesh method is similarly straight forward. Steps to import automatically into Unreal are:

1. Open file of choice and convert to a CSV.
2. Use Open3D to convert the point cloud to a mesh.
3. Import the static mesh into Unreal Engine using the "Content Drawer".

To spawn the bathymetric mesh in Unreal, one can manually drag the bathymetric mesh into the world. For automated inclusion at game time, a new Actor is created with a UStaticMeshComponent. Selection between meshes is enabled using INI configuration and a map of available bathymetries.

Sensor Integration

Sensors requiring access to the meshes are created as components within Unreal Engine and added to vehicle assets as they would exist on a physical vehicle. The method taken for creating such sensors was to first create a new C++ component class and create a pointer to a ROS2 handler implemented in the engine. This handler allows the component to become both publisher and subscriber as needed to handle data operations. In addition, each implementation can be used for multiple hardware devices of the same type. For example, many Doppler Velocity Logs (DVLs) will require the same raw data from the visualization, but have varying ranges, accuracies, and errors. As such, the implementation of the sensor within the visualization is largely agnostic to the DVL being simulated. Rather than including all information into each subcomponent, it is instead a basic configuration with four ray traces that intersect the bathymetry or other objects in the environment and return a range. This design choice allows for testing vehicles with differing sensor brands or models without requiring changes in the Unreal asset. The recorded ranges are then communicated to the ROS2 DVL implementation within the simulator backend, which handles the raw data, injection of faults, maximum range, accuracy and precision limitations, and converts the data into the format of a given sensor for communication with the vehicle platform. Visualizing DVL beams is done using ray traces within Unreal, with the traces having an option to use highlights for visualization. While this visualization requires minimal code, the ability to visually determine where the beams contact the world is beneficial in troubleshooting many types of unwanted vehicle behavior.

VEHICLE MODEL

Simulated vehicles are comprised of a federation of models that, like the complex system, result in an overall vehicle representation. A UUV model includes input physical properties (volume, mass, inertia, etc.), the hydrodynamic signature of the hull (maneuvering coefficients), actuator models, and sensor models. These aspects of the vehicle model play a significant role when calculating the updated vehicle state over time within the simulation's physics solver.

Physics Solver Formulation

The physics engine is the core of the simulation environment and is responsible for solving the dynamic equations of motion that describe a vehicle's response both underwater and on the surface. The physics solver is its own ROS2 node that computes the effects of body-relative forces and moments on a vehicle and then outputs those states in an inertial reference frame. The physics engine assumes rigid body kinematics and requires data from the following:

1. Vehicle hydrodynamic forces and moments
2. Vehicle hydrostatic forces and moments
3. Vehicle actuator forces and moments
4. Environmental forces

There are models that describe the vehicle under simulation (for hydrodynamics and hydrostatics), the vehicle control surfaces and propulsor, and the environment (ocean density, ocean currents, obstacles and terrain, sea state effects, and surface effects) that are detailed in follow-on sections.

The physics engine is a 6-degree-of-freedom maneuvering solver developed in accordance with torpedo theory formulated by (Fossen, 2011). To solve the motion of an underwater vehicle, all six degrees of freedom need to be modeled. The nonlinear equations of motion are expressed in the body fixed reference frame and presented in vectorial representation (Fossen, 2011) are detailed below. Assumptions have been applied that no ballast tanks are present, the vehicle is operating at depths out of the wind effected zone, and wave effects are implemented via modifications to water velocities (detailed in later section). The resulting equations of motion with velocities solved in the body reference frame using Eulerian representation are:

$$\dot{\eta} = [J_\theta(\eta)]\nu \qquad (1)$$

$$[M]\dot{\nu} + [C(\nu)]\nu + [D(\nu)]\nu + g(\eta) = \tau \qquad (2)$$

Where:

$\eta = [x, y, z, \phi, \theta, \psi]^\intercal$ = position and orientation of the vehicle
$\nu = [u, v, w, p, q, r]^\intercal$ = linear and angular velocities of the vehicle
$M = M_{RB} + M_A$ = system inertia matrix (RB: rigid body, A: added mass)
$C(\nu) = C_{RB}(\nu) + C_A(\nu)$ = Coriolis-centripetal matrix (RB: rigid body, A: hydrodynamic/added mass)
$D = D + D_\eta(\nu)$ = Damping matrix
$g(\eta)$ = vector of generalized gravitational and buoyancy forces
τ = vector of total forces and moments on the vehicle
$J_\theta(\eta)$ = transformation matrix for Eulerian representation

Hydrodynamic coefficients representing the vehicle hull, forces and moments resulting from actuated components, and environmental flow effects are components of the variables represented in Equation 2.

Hydrodynamic Coefficients (Vehicle Profile)

Maneuvering theory is assumed for the derivation of hydrodynamic coefficients where frequency-dependent added mass (M_A) and potential damping ($D(\nu)$) are approximated by constant values. The assumption is made that hydrodynamic forces and moments can be approximated at one frequency of oscillation such that fluid memory effects necessary in seakeeping theory can be neglected. Derivations for some coefficients for a symmetrical UUV profile are defined in (Prestero, 2001) based on empirical formulation, strip theory, and geometric approximations. More commonly, values are determined for these coefficients by using CFD simulation data using similar methods to (Njaka, Brizzolara, & Stillwell, 2022).

Actuator Modeling

Actuators as defined within the simulator include both propeller and control fins, with 3-fin and 4-fin (both X and +) configurations supported. This is achieved through the addition of a motor controller node fed by separate fin and propeller nodes. As described with the DVL in the *Sensor Integration* section, this allows for running different combinations of controllers, fins, and propellers.

Propeller Modeling
Propeller models are implemented into simulation through various methods. Low fidelity models include the use of physical static testing to develop and fit thrust vs rotation per minute (RPM) curves. In many cases, this may be sufficient and can be adequately represented with a second order polynomial. Higher fidelity models include the use of coefficient characterization. This is done through obtaining advanced coefficients for a given propeller using CFD at various RPM values and plotting both thrust and torque coefficient data to determine a best fit function.

Fin Modeling
A control fin will not experience the same flow velocities as experienced by the body and the overall flow velocity is similarly unlikely to be directly parallel to the body. As a result, to accurately compute lift and drag produced by a control fin an effective angle of attack is computed. For vehicles with a cruciform fin configuration, the effective fin angle is found using the following equations in body fixed coordinates.

$$\delta_{re} = \delta_r - \beta_{re} \qquad (3)$$

$$\delta_{se} = \delta_s + \beta_{se} \qquad (4)$$

Where:

δ_{re} = the effective rudder fin angle
δ_{se} = the effective pitch fin angle
δ_r = the actuated rudder fin angle
δ_s = the actuated pitch fin angle
β_{re} = the angle attack of the rudder zero plane
β_{se} = the angle attack of the pitch fin zero plane

In addition to modeling true fin angle of attack, empirical modeling of fin throw rates and error of fin throw angle was conducted and implemented to increase fin model fidelity and improve vehicle dynamic response. These effects were characterized using an oscilloscope on the actual hardware.

Sensor Modeling

Sensor modeling differs by sensor type and available information. All sensor models begin with identification of required information from either the virtual world or physics solver as well as the sensor interface control document (ICD) to determine communication standards. Continuing with the example above of a DVL, the required external information needed are vehicle velocity calculated by the physics solver, and beam lengths returned by the virtual world platform. From the ICD the message formats are determined, as well as any commands that need to be made available. Information from the simulator is received using ROS2 subscriptions which update class variables. Within each sensor's main process loop, a receive buffer is read and parsed to determine any commands received from the vehicle. Actions taken by the sensor are determined based on information from sensor ICDs, user manuals, and, whenever possible, directly observed responses from the physical sensor. If a sensor sends data at a specific rate when active, this will occur after action is taken based on vehicle input.

Once a baseline sensor function is implemented and tested with the vehicle, additional fidelity can be added. The first addition is sensor accuracy, precision, and other known metrics such as range and drift rate. Further development includes the ability to generate sensor faults either on command or based on environmental conditions. Low fidelity implementations are characterized as supplying the minimum data and communication necessary for the simulated sensor to remain largely transparent to the vehicle. Medium and high-fidelity models introduce sensor parameters, faults, and modify functionality of a given sensor based on configuration commands. For a DVL, the low fidelity model must transmit a startup message as well as periodically send a message with beam altitudes populated. Higher fidelity models include maximum beam ranges and accuracy values based on the water column data, loss of bottom lock, sensor faults, and loss of communication.

Flow Effects

Ocean effects are exerted on the vehicle by asserting the assumptions of irrotational flow, allowing the equations of motion to be expressed with respect to the relative velocity vector (Fossen, 2011):

$$\nu_r = \nu - \nu_c - \nu_w - \nu_t \tag{5}$$

$$\underbrace{[M_{RB}]\dot{\nu_r} + [C_{RB}(\nu_r)]\nu_r + g(\eta)}_{rigid-body\ and\ hydrostatic} + \underbrace{[M_A]\dot{\nu_r} + [C_A(\nu_r)]\nu_r + [D(\nu_r)]\nu_r}_{hydrodynamic} = \tau \tag{6}$$

Where:

$\nu_c = [u_c, v_c, w_c, 0, 0, 0]^\intercal$, uniform ocean current vector
$\nu_w = [u_w, v_w, w_w, 0, 0, 0]^\intercal$, vector of velocity influences from wave effects
$\nu_t = [u_t, v_t, w_t, 0, 0, 0]^\intercal$, vector of velocity influences from ocean turbulence
(Drennan, Donelan, & Katsaros, 1996)

The relative velocity representation decomposes the effects on velocity from the various flow effects: current, waves, and turbulence. These effects are incorporated in the physics solver based on input state data from environment models.

Water Profile

The temperature and salinity of sea water change with location (Atlantic Ocean or Pacific Ocean), time of year (Summer or Winter), and depth, thus effecting the density of the water in $g(\eta)$. Temperature and salinity profiles were extracted from literature (Allmendinger, 1990) and implemented into the solver as piecewise best-fit functions fit to the data. These values are calculated based on the depth of the vehicle and used to calculate the density of the water based on (Gill, 1982). All models that use density in the simulation environment subscribe to the output of this water profile model. This implementation can be extended for custom profiles from measured environmental data as well.

Ocean Currents

Ocean currents are modeled as uniform in magnitude and direction in the simulation environment populating ν_c in Equation 5. By default, current is set to 0 but can be populated during input configuration to perform trade space studies of its effects on vehicle operations to inform mission planning. The uniform formulation is simple but allows for a controlled analysis of the vehicle's response to these additional flow contributions.

Wave Effects

The presence of waves contributes additional forces on the vehicle throughout the mission, particularly when at the surface or at shallow depths. In many cases, these forces can have a large effect on the performance of the vehicle to operate its objectives. Wave effects are modeled as irregular seas and are incorporated as a fluid velocity contribution experienced by the vehicle, ν_w, as mentioned in Equation 5. Required inputs for the wave effects model include number of components for the wave spectrum (n), general wave direction (ψ_w), sea state code (1-8), wave spectrum (Brettschneider, Ochi-Hubble, Pierson Moskowitz, or JONSWAP), and angular frequency range for the spectrum. Formulation from (Faltinsen O. M., 2005) is applied based on irregular wave theory and the recorded peak period and maximum wave height for each sea state mode.

Integrating CFD Flow Field into the Virtual World Platform

In addition to the built-in subprocesses for flow effects previously mentioned, the simulator can import fluid flow data previously solved for by external CFD programs and impart forces from that flow data on the affected vehicle. This capability is critical for modeling the environment in applications where the flow in the simulated area of operation is nonuniform. An example application is maneuvering around a large or complex submerged object subjected to an ocean current; the flow around the object could be modeled using CFD software and then imported into the virtual environment. These resulting nonuniform flow conditions are not captured by the maneuvering solver described in the previous sections. The CFD import capability outlined in following subsections provide an introductory step to bridge the gap in the ability to capturing the vehicles response in nonuniform flow conditions in the existing real time simulator. This implementation can be used for an initial design of experiments assessment. Full CFD simulations should be completed if precision is needed for evaluating the response of the vehicle in these complex flow fields.

Data Storage & Search

Since the virtual environment runs in real-time, accessing flow field data quickly is critical. The R*-tree data structure (Beckmann, 1990) satisfies this requirement, with a one-time data loading time complexity of $O(n\, log\, n)$ and a search complexity of $O(log\, n)$ where n is the number of points in the field. Every member of the tree contains a position and velocity vector. Search is completed with K-nearest neighbors in combination with a radial distance cutoff. Loading of time-varying CFD data is supported, with each timestep stored efficiently with $O(n)$ space complexity as a separate R*-tree (Rstar Crate Team, n.d.).

Velocity Interpolation

Interpolation of velocity values within the flow field is achieved using an Inverse Distance Weighting (IDW) method for spatial data, based on Shepard's method (Shephard, 1968). IDW is used to interpolate each component of velocity separately. Interpolated velocities are integrated into the physics solver by similar method to the addition of the ocean current ν_c mentioned above. This interpolation method was tested using internally generated CFD data imported from OpenFOAM (OpenCFD Ltd, 2024). The CFD cases include the well-studied DARPA SUBOFF submarine (Groves, Huang, & Chang, 1989), flow around a sphere, and uniform steady flow. The geometry used for SUBOFF was at a 0 angle of attack with an inlet speed of 4 knots. Randomly sampling across subsets of the domains and comparing interpolated cell velocities with CFD cell velocities yielded a mean percent error of less than 1%, which is reasonable for capturing realistic flow effects.

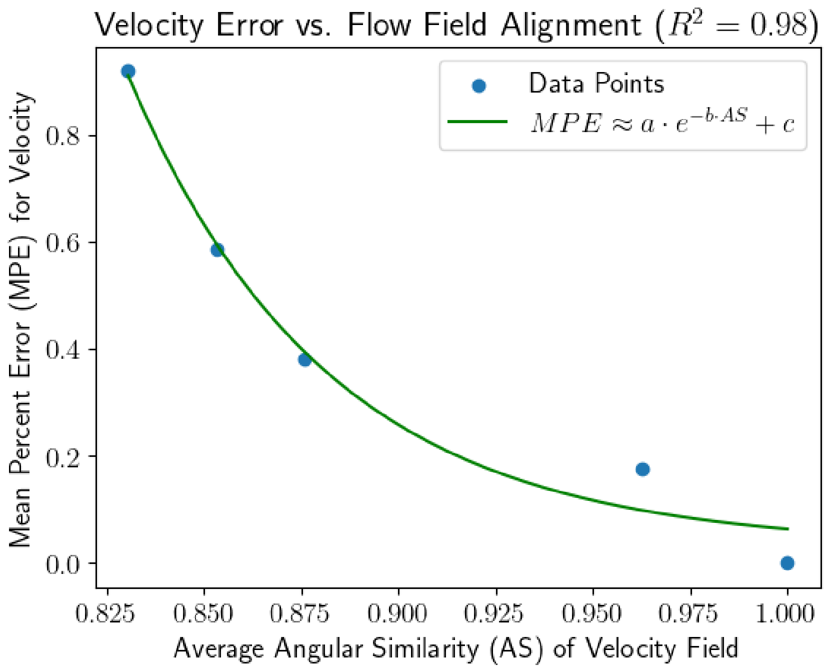

Figure 3: Velocity Interpolation Validation

The plot in Figure 3 displays a clear negative relationship between the uniformity of flow and the interpolation error for velocity. Angular similarity (AS) between two vectors $\vec{A}, \vec{B}$ ranges from 0 to 1 and is defined as $1 - \frac{1}{\pi}\cos^{-1}\frac{\vec{A}\cdot\vec{B}}{\|A\|\|B\|}$. Percent error for velocity $\vec{U}$ is defined as $100 \cdot \frac{\|\vec{U}_{CFD} - \vec{U}_{INTERP}\|}{\|\vec{U}_{CFD}\|}$. The coefficients of the exponential fit shown in the legend are $a \approx 951 \times 10^4$, $b \approx 19.5$, and $c \approx 0.03$. The converging trend verifies accuracy of the interpolation method as implemented in the simulation environment.

Example Parameter Study

Data from the same DARPA SUBOFF test case was used to demonstrate the flexibility of the implementation in that the number of nearby CFD data points used for interpolation can be tuned for specific applications. The number of neighboring points (K) relates closely to the accuracy of interpolation. Similarly, the search radius for nearby points is linked to the physical size of the simulated vehicle and the minimum size of the fluid structures needed for the application at hand. By varying search radius and K at random points in time and space it is possible to construct curves (Figure 4) showing the effect that increasing K has on interpolation error aggregated across search radii. Observing the resulting trends in the error allows for selection of an optimal value of K to balance runtime cost and interpolation error while using a radius that is physically meaningful to the application. For instance, in the example shown in Figure 4, if a radius of 0.1m or less is desired for the application, a radius of 0.05m with a K between 20 and 50 would be appropriate to reduce error while balancing acceptable computational costs. In this case, K values less than 20 do not appear to follow a reliable trend.

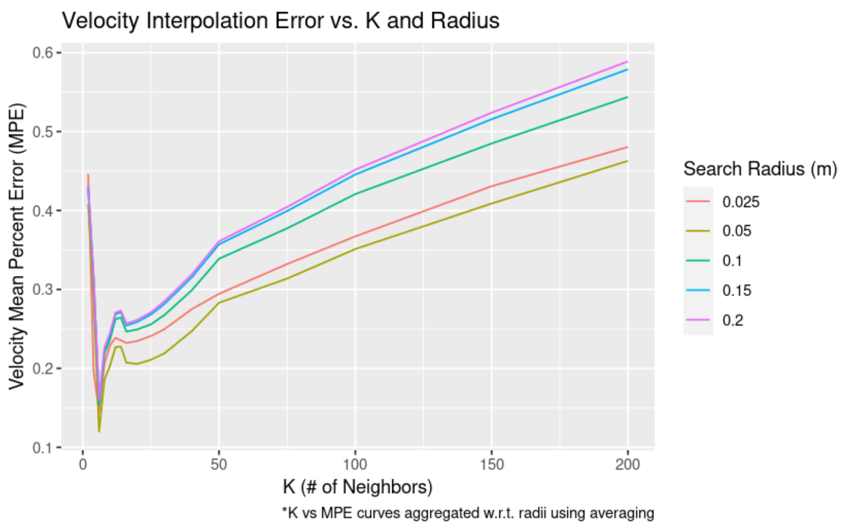

Figure 4: Interpolation Error vs. Flow Field Neighbor Search Parameters

CONCLUSION

The simulator highlighted within this paper is a capable and robust testbed for software and hardware systems. This system places both whole and partial vehicles into a simulated environment capable of exercising vehicle software for REMUS vehicles. Due to the nature of the architecture (Figure 2), the simulator is not limited exclusively to the REMUS platform and is able to exercise any underwater vehicle given accurate vehicle parameters and knowledge of sensor systems. Expansion of new sensors and sensor types is implemented using factory implementations or new nodes, with ROS2 handling data transfer between the visualization and backend as well as any required data for a given sensor implementation.

The addition of Unreal Engine as a visualization platform further enhances the simulator capability to include both real and synthetic bathymetry. Multiple NOAA maps can be combined for use with longer missions or those spanning across survey sites. The visualization platform also allows for three-dimensional playback of existing simulations for diagnostics and testing.

The simulator provides a robust framework for continued growth, with future work primarily directed at increasing the capability of the virtual world platform. Future developments are also planned in the area of CFD integration, specifically more accurate interpolation methods and handling of complex flows. The current implementation is best suited for when local regions in the domain contain approximately laminar flows without large fluid velocity gradients. Future improvements to the visualization platform include the addition of collisions into the simulator, new sensor types, and synthetic bathymetry tooling. Current sensors of interest include forward looking, side scan, and gap filler sonar systems and cameras. Methods of importing bathymetry into the simulator has been used for synthetic bathymetries, as well as adding targets on the seafloor. However, future work into synthetic bathymetry and coastal topography generation is anticipated to increase usability for those unfamiliar with the existing simulator.

ACKNOWLEDGEMENTS

The authors would like to acknowledge their colleagues at HII in the Modeling and Simulation development team: Madeline Aitkenhead, Matthew Guenther, Brendan Martin, Clayton Spaulding, and Timothy Swierzewski. Working in a team that is enthusiastic about the research, dedicated to a quality product, and fully committed to putting their work into action is motivating and inspirational.

REFERENCES

Allmendinger, E. (1990). Submersible Vehicle Systems Design. Jersey City, NJ: Society of Naval Architects and Marine Engineers (SNAME).

Beckmann, N. K.-P. (1990). The R*-tree: an efficient and robust access method for points and rectangles. *SIGMOD '90: Proceedings of the 1990 ACM SIGMOD international conference on Management of data.* Atlantic City New Jersey USA: Association for Computing Machinery.

Drennan, W. M., Donelan, M. A., & Katsaros, K. B. (1996). Oceanic Turbulence Dissipation Measurements in SWADE. *Journal of Physical Oceanography*, 26.

Epic Games. (2024). *Unreal Engine 5*. Retrieved from unrealengine.com: https://www.unrealengine.com/en-US/unreal-engine-5

Faltinsen, O. M. (2005). *Hydrodynamics of High-Speed Marine Vehicles.* New York, NY: Cambridge University Press.

Faltinsen, O. M. (2005). *Hydrodynamics of High-Speed Marine Vehicles.* New York, NY: Cambridge University Press.

Fossen, T. I. (2011). *Handbook of Marine Craft Hydrodynamics and Motion Control.* Trondheim, Norway: Wiley.

Gill, A. E. (1982). *Atmosphere-Ocean Dynamics.* San Deigo, CA: Academic Press.

Groves, N. C., Huang, T. T., & Chang, M. S. (1989). *Geometric Characteristics of DARPA (Defense Advanced Research Projects Agency) SUBOFF Models (DTRC Model Numbers 5470 and 5471).* DAVID TAYLOR RESEARCH CENTER BETHESDA MD SHIP HYDROMECHANICS DEPT.

Njaka, T., Brizzolara, S., & Stillwell, D. (2022). Guide for CFD Informed AUV Maneuvering Models. *SNAME T&R Bulletin 1-45*. Alexandria, VA: The Society for Naval Architects and Marine Engineers.

NOAA. (2024). *Bathymetric Data Viewer*. Retrieved from ncei.noaa.gov: https://www.ncei.noaa.gov/maps/bathymetry/

Open Robotics. (2024). *ROS2 Documentation*. Retrieved from ROS2 Documentation: Foxy: https://docs.ros.org/en/foxy/index.html

Open3D. (2024). *Open3D*. Retrieved from Open3D: https://www.open3d.org/

OpenCFD Ltd. (2024). *About OpenFOAM*. Retrieved from openfoam.com: https://www.openfoam.com/

Prestero, T. (2001, September). Verification of a Six-Degree of Freedom Simulation Model for the REMUS Autonomous Underwater Vehicle. Cambridge, MA: Massachusetts Institute of Technology, MS Thesis.

Rstar Crate Team. (n.d.). *Struct rstar::RTree*. Retrieved from docs.rs: https://docs.rs/rstar/latest/rstar/struct.RTree.html

Shephard, D. (1968). A two-dimensional interpolation function for irregularly-spaced data. *ACM '68: Proceedings of the 1968 23rd ACM national conference.* Association for Computing Machinery.

Warmerdam, F., & Rouault, E. (2024). *GDAL*. Retrieved from GDAL: https://gdal.org/index.html