A Simulator for Underwater Human-Robot Interaction Scenarios

Kevin J. DeMarco
Georgia Institute of Technology
Electrical and Computer Engineering
Atlanta, United States
demarco@gatech.edu

Michael E. West
Georgia Institute of Technology
Georgia Tech Research Institute
Atlanta, United States
mick.west@gtri.gatech.edu

Ayanna M. Howard
Georgia Institute of Technology
Electrical and Computer Engineering
Atlanta, United States
ayanna.howard@ece.gatech.edu

***Abstract*—Divers work in dangerous environments that place severe constraints on the types of activities that divers can accomplish. The development of a underwater robotic assistant may help a human diver accomplish tasks more efficiently and safer. However, before Underwater Human-Robot Interaction (UHRI) can be deployed in the field, the UHRI algorithms must be tested and validated in a simulator. In order to test future UHRI algorithms and behaviors, an underwater simulator, based on the Modular OpenRobots Simulation Engine (MORSE) was developed. The Mission Oriented Operating Suite (MOOS) autonomy framework was integrated with the Robot Operating System (ROS) and MORSE to demonstrate a basic UHRI scenario: the VideoRay Pro 4 Remotely Operated Vehicle (ROV) trailing a human diver. The UHRI simulator makes use of other open source projects to enable the programmer to easily incorporate new 3D models into MORSE and adjust the fidelity of sensor and motion models based on scenario goals.**

Keywords*—*Underwater Human-Robot Interaction, Robotics Simulator, MORSE, 2D Imaging Sonar

I. Introduction

Both commercial and military divers operate in harsh, unforgiving environments. Commercial divers are often required to remain at-depth for long periods of time and can only resurface after completing a complex sequence of decompression stages in order to avoid decompression sickness. Similarly, military divers are required to complete a number of complex tasks including the fabrication of underwater structures and repairing underwater cables. Navy Explosive Ordnance Disposal Divers are also required to work with extremely hazardous material. In addition to monitoring their own life support systems, divers are also faced with the challenge of having to conduct operations in near-zero visibility conditions due to water depth and turbidity [1]. Robotic assistants have shown great promise in space environments [2]. This research builds upon this tremendous robotic assistant capability within the unstructured underwater environment. However, there are a number of machine vision and autonomy challenges posed in this complex environment that must be addressed before an effective assistant is deployed.

An effective underwater robotic assistant will have to be able to process both optical imagery and high-frequency sonar imagery in order to accurately assess the environment and interact with the diver. In [3], two-dimensional computer vision techniques were applied to real sonar imagery in order to detect and track a human diver. The sonar data was acquired with a BlueView P900 2D imaging sonar attached to a VideoRay Pro 4 as shown in Figure 1.

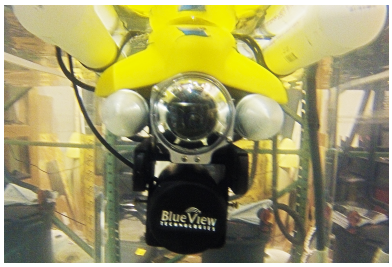


Fig. 1: VideoRay Pro 4 with attached BlueView P900 in water tank.

Through the initial study, fifty sonar data sets of a human diver moving through the water were acquired. In order to decrease autonomy development time, a means to simulate a greater variety of situations and environments was required. Thus, the next step in the development of the underwater robotic assistant was the implementation of a 2D imaging sonar simulation. This simulation allows the developer to process simulated sonar and optical imagery and test various autonomy algorithms to command the motion of the autonomous underwater vehicle (AUV) or remotely operated vehicle (ROV). In addition, a model of a simulated diver was developed in order to test various Underwater Human-Robot Interaction (UHRI) scenarios.

II. Related Work

The modeling of high frequency sonar systems has been a topic of research for numerous researchers in academia and industry. Early computer simulations of sonar often focused on synthetic side scan sonar models and the sonar images could only be visualized after the simulation was complete. This was the case in [4], where the authors also discussed the difficulties of comparing simulated results to actual sonar data. The authors of [4] utilized ray tracing in order to generate their sonar images, which was also discussed in [5]. The process of ray tracing involves advancing a simulated ray out of the sonar transmitter until the ray comes in contact with an object or boundary. If the temperature of the water is assumed to be constant, the ray is an adequate approximation

978-0-933957-40-4 ©2013 MTS

of a sonar beam. This is due to the fact that if the water temperature changes along the path of the sonar beam, the beam will bend. The process of projecting rays is repeated for an array until a three dimensional point cloud is constructed from the ray detection points. The authors of [5] noted that ray tracing can provide highly accurate sonar images; however, ray tracing is a computationally expensive process. More recently, researchers have been looking for alternatives to ray tracing and image rasterization, which has led to a process called tube tracing [6]. Tube tracing involves using multiple rays to form a polygon or footprint on a detected boundary. Tube tracing has the advantage of being less computationally expensive compared to ray tracing and the tube volume is more characteristic of sonar beams compared to ideal rays. The capability of tube tracing was shown in Guriot for forward-looking sonar simulations [7]. Much of the same theory that applied to side scan sonar simulations also applied to forward-looking sonar simulations, but his research also focused on synthetically generating realistic sea floor images with the use of micro-textures. While previous research alluded to the computational complexity, none provided a specific metric by which to measure the computational complexity or evaluate the simulation time. This is mostly due to the fact that most of the simulations in the literature were not close to real-time.

The literature on UHRI is not nearly as extensive as the literature on the modeling of sonar systems. Most of the previous work on UHRI has focused on enabling track-and-trail behaviors for an autonomous agent through the detection and tracking of a cooperative diver. For example, in [8], researchers used the frequency at which the diver's fins undulated as a detection feature. Sattar and Dudek also published research on the use of bar-codes and a language they developed called *RobotChat* that enabled a human diver to command an underwater robot through its optical camera [9]. Finally, in [10], the authors used an ROV's optical camera and gestures from a diver to implement basic human-robot interaction. One of the main issues with the previously discussed works is that they relied heavily upon an optical camera for interpreting the environment and interacting with the diver. Unfortunately, in the underwater environment, the turbidity of the water renders most optical detection of limited use past even a few feet, even if the vehicle is equipped with a quality light source. An additional sensor that can measure range and bearing to a target, such as a 2D imaging sonar, would tremendously improve an ROV's ability to detect and track a cooperating human diver.

III. Simulation Engine Requirements and Selection

There are a number of 3D simulation engines for robotics research and development. Three of the most popular simulation environments are Player/Stage/Gazebo [11] [12], the Microsoft Robotics Developer Studio (MRDS) [13], and the Urban Search And Rescue Simulator (USARSim) [14]. There are countless other simulation engines that rely upon Bullet Physics [15] or Open Dynamics Engine (ODE) [16] for the simulator's physics engine and Open Scene Graph [17] for the simulator's visualization engine. In the selection of an appropriate simulation engine, the main requirements are:

- The simulator must provide hooks to common robotics middleware architectures (i.e. ROS, MOOS, etc.)
- It should be simple to build or add new 3D objects
- The simulator should provide a means to script events external to the target autonomous agent
- There must be a means for adding dynamical models to the physics engine

After comparing various robotics simulators, such as Player/Stage/Gazebo, the Microsoft Robotics Developer Studio (MRDS), and the Urban Search and Rescue Simulator (USARSim), it was concluded that the Modular OpenRobots Simulation Engine (MORSE) provided the most modern robotics simulation environment that was highly customizable [18].

A. The Modular OpenRobots Simulation Engine

MORSE was initially developed at LAAS-CNRS, a public French robotics laboratory, and heavily utilizes the popular open source game engine and 3D visualizer, Blender [18]. Researchers have already developed robotics simulations with ground, water surface, and aerial robots in MORSE that utilize simulated SICK lasers, video cameras, GPS units, IMUs, depth cameras, etc. Since MORSE sits on top of Blender, it uses the Bullet Physics game engine. Finally, MORSE has already been adapted to work with several popular middleware architectures, such as The Robot Operating System (ROS), the Mission Oriented Operating Suite (MOOS), Yet Another Robot Platform (YARP) [19], and Pocolibs [20].

The most stable and extensible method of communicating data between the MORSE simulator and an external process is with the use of ROS. Thus, a new ROS package entitled, "videoray," was created to encompass all of the individual processes that were used to simulate the VideoRay ROV's dynamics model, controller, and associated planning engines. Previously, a MOOS/ROS Bridge program was developed in [21] that shuffled data between a MOOS Community and a ROS Core based on a simple XML configuration file. Figure 2 shows how the MOOS/ROS Bridge sits between MOOS and ROS in order to pass information concerning desired speed, heading, and depth from the MOOS side to the ROS side.

Likewise, the ROV's pose and the human's pose are transferred from the ROS side to the MOOS side, so that the IvPHelm engine can generate a desired trajectory. An important aspect of the interface between MORSE and ROS is the type of information that is transferred between the two components. Internally, Blender performs collision detection, which affects the final poses of obstacles in the simulator. Also, Blender is capable of integrating linear and angular velocities in order to determine an object's pose. Since the VideoRay dynamics model in ROS does not have knowledge of obstacles in the World Model, it is not capable of performing collision detection, which could affect the VideoRay's final pose. Instead, Blender receives the velocity information, integrates the model, performs collision detection, and then feeds the full state model back to the VideoRay dynamics model for the next iteration in the simulation.

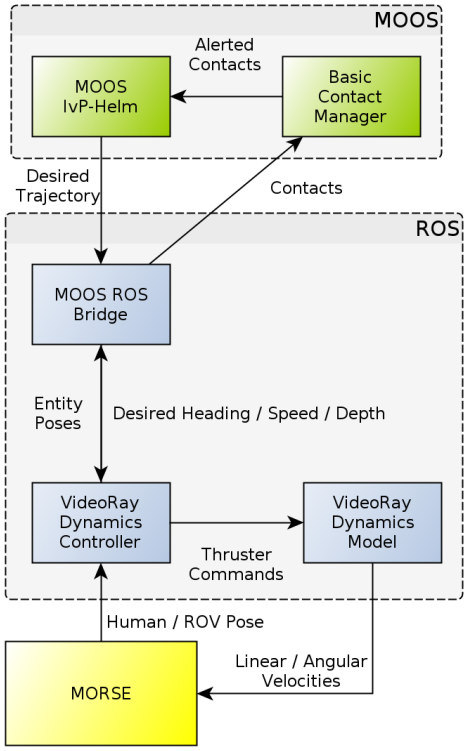


Fig. 2: System configuration with MOOS/ROS Bridge.

IV. Remotely Operated Vehicle Model

The VideoRay Pro 4 was selected as the remotely operated vehicle (ROV) for the UHRI simulation due to its accessibility in price and ease-of-use in terms of mission deployment and hardware maintenance. The actual VideoRay Pro 4 system consists of an ROV that communicates with the top-side base station via a tether that contains an RS-485 serial data link. While a Windows-based desktop application is provided by VideoRay, in order to drive the VideoRay with autonomy modules in ROS and MOOS-IvP, a Linux-based VideoRay application was developed. The Linux-based control station consists of MOOS-IvP's pMarineViewer application and several back-end applications that convert IvPHelm's desired trajectory commands into serial RS-485 commands that directly drive the VideoRay's thrusters. It is important to understand how the actual VideoRay can be integrated with ROS and MOOS-IvP based on its hardware characteristics, so that the code-base can be shared between the simulation environment and the actual mission deployment environment.

A. VideoRay Pro 4 3D Model

The 3D model of the VideoRay Pro 4 was initially received as a SolidWorks file. Within SolidWorks, the 3D model was exported as a Virtual Reality Modeling Language (VRML) file, which was imported into Blender. Bounding boxes were added to the Blender model for collision detection, the model was scaled to fit appropriately within the simulation environment, and the 3D model was aligned within the coordinate frame to coincide with MORSE's coordinate frame. The final result is provided in Figure 3, which displays the Blender quad view of

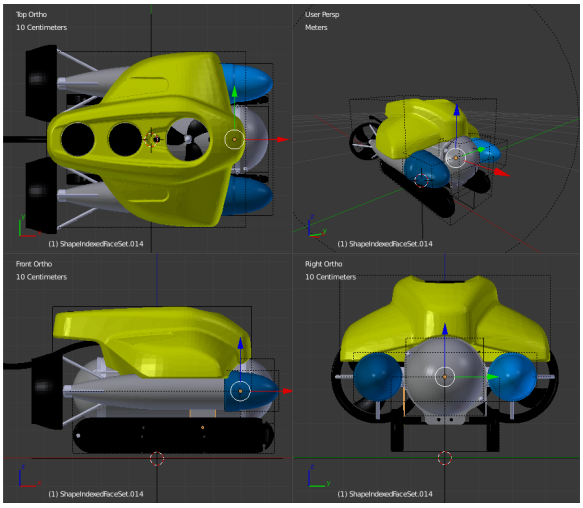


Fig. 3: Quad view of the VideoRay Pro 4 3D model.

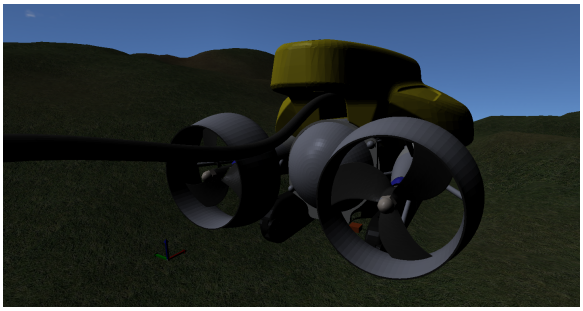


Fig. 4: VideoRay Pro 4 3D model in the MORSE simulator.

the VideoRay Pro 4. A screen capture of the VideoRay's final rendering within the MORSE simulator is shown in Figure 4.

B. VideoRay Pro 4 Dynamics Model

The kinematic model that was implemented in the simulator for the VideoRay ROV was heavily based upon the general 6 DOF model described by Fossen [22] and the hydrodynamic and inertial coefficients that were determined analytically and empirically in [23]. Since the author of [23] assumed planes of symmetry in the port-starboard and top-bottom planes, the equations of motion simplified to the following in the horizontal plane.

$$m_{11}\dot{u} = -m_{22}vr + X_u u + X_{u|u|}u|u| + X \quad (1)$$

$$m_{22}\dot{v} = m_{11}ur + Y_v v + Y_{v|v|}v|v| \quad (2)$$

$$J\dot{r} = N_r r + N_{r|r|}r|r| + N \quad (3)$$

The Z-direction motion is decoupled from the horizontal motion and is described by the following equation:

$$m_{33}\dot{w} = Z_w w + Z_{w|w|}w|w| + Z \quad (4)$$

The equations of motion were simulated in the GNU Octave environment by enabling the thrusters for a period of time and

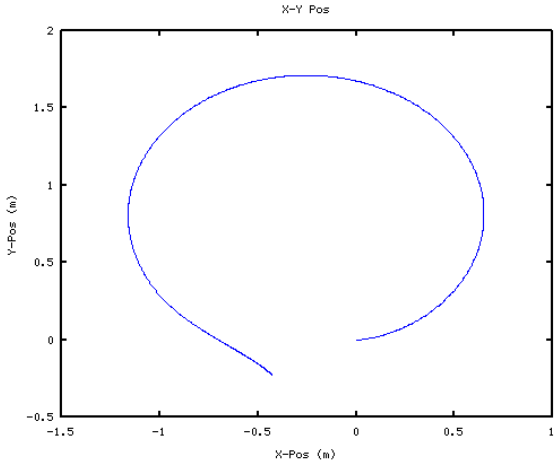


Fig. 5: X-Y Plot of VideoRay's trajectory.

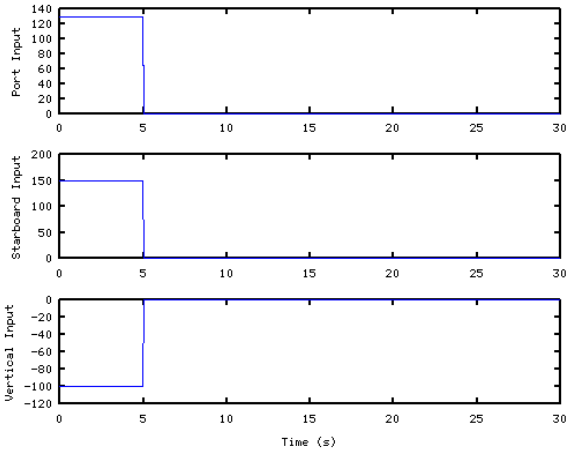


Fig. 6: Thruster input.

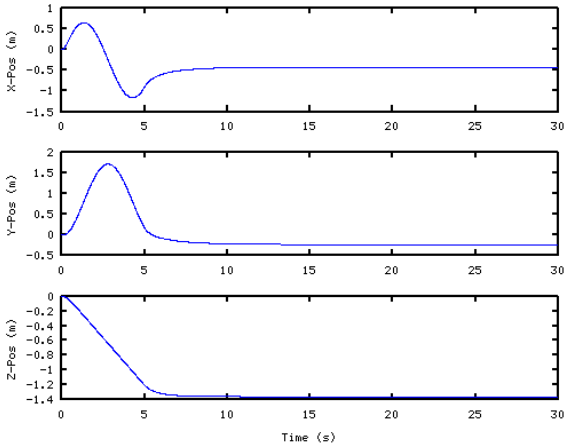


Fig. 7: VideoRay's position variables.

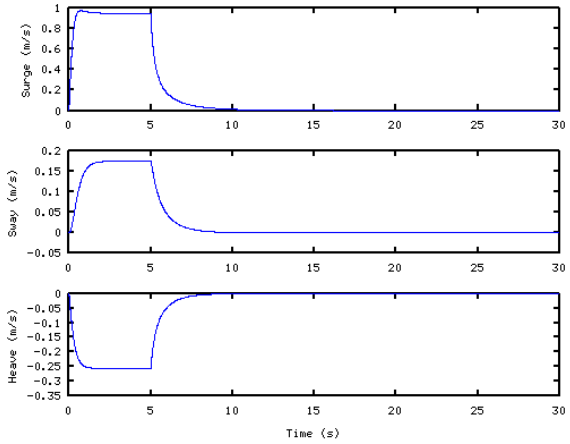


Fig. 8: VideoRay's linear velocity variables.

then disabling the thrusters in order to validate that the system variables settle back to zero. The input for each thruster is in the range -150 to 150, where the input is a dimensionless unit that generates a proportional change in thruster rotational speed. Outside the range, the thrusters are saturated. In this simulation, the starboard thruster was provided an input of 150, the port thruster was provided an input of 130, and the vertical thruster was provided an input of -100. This results in the ROV moving in the negative Z-direction while moving in a circular trajectory in the counter-clockwise direction. The X-Y plot of the ROV's top-down trajectory is given in Figure 5. A plot of the thruster input is given in Figure 6. Since the thruster input is disabled after five seconds, the tail end of the X-Y trajectory shows that the ROV is no longer rotating as it decelerates to a stop. A plot of all position variables in Figure 7, demonstrate that the ROV is increasing in depth. The linear velocities of the model are plotted in Figure 8, where the deceleration due to hydrodynamic damping is prominent.

After the open loop simulation was verified in the GNU Octave simulation, the VideoRay's kinematic model was implemented in C++ for integration with the Mission Oriented Operating Suite (MOOS) [24], the Robot Operating System (ROS) [25], and MORSE. The C++ model implementation was developed with the use of Boost's `odeint` package [26]. The `odeint` package eased the porting of code from Octave to C++ because the model definition and differential solver in `odeint` are very similar to those found in Matlab and Octave. Thus, the Octave code was essentially copied from an Octave script and pasted into a C++ file with minor changes to variable types and function calls.

C. Proportional Control of VideoRay

The first controller developed was a simple proportional controller that merely compared a desired parameter of the ROV's state to the actual value. Currently, a depth controller, a heading controller, and a speed controller have been developed. What is interesting about the heading and speed controllers is that they can be in conflict with each other since the forward speed is controlled by the horizontal thrusters' force, but the heading control is also controlled by the horizontal thrusters. Thus, a control law was developed where the final output of each thruster is a weighted sum of the output of the heading and speed controllers. For example, if the heading and speed

errors are defined by

$$e_{heading} = desired\_heading - actual\_heading \quad (5)$$
$$e_{speed} = desired\_speed - actual\_speed \quad (6)$$

and the control law for the port-side thruster is

$$u_{port\_heading} = K_p e_{heading} \quad (7)$$
$$u_{port\_speed} = K_p e_{speed} \quad (8)$$

The final control law for the port-side thruster becomes

$$u_{port} = w_{heading} u_{port\_heading} + w_{speed} u_{port\_speed} \quad (9)$$

Currently, equal weighting is given to both the heading and speed controllers; thus, both the weights are set to 0.5. For the purposes of this simulation, this controller provides adequate results.

V. Forward-Looking Sonar Model

While robotics simulators support planar sonar sensors, there are not any open source implementations of 2D imaging sonars, such as the forward-looking sonars provided by BlueView [27]. An example of one of the imaging sonars provided by BlueView is shown in Figure 9(a). The imaging sonar produces an image similar to the image shown in Figure 9(b), where the angle of the sonar sweep is dependent upon the type of sonar. The features near the bottom of the image are closest to the sonar, while the features near the top of the image are farthest from the sonar. The specifications listed in Table I were obtained from BlueView's website and were used for the development of the sonar model [27].

(a) BlueView P900 series. (b) 2D imaging sonar output.

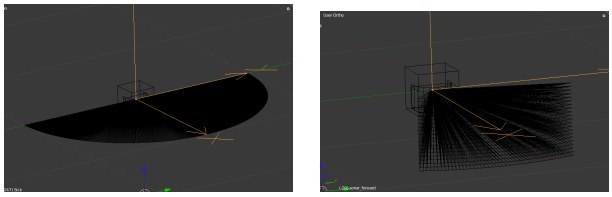
Fig. 9: BlueView 2D imaging sonar.

TABLE I: Forward-Looking Sonar Model Specifications.

Operating Frequency	900 kHz
Update Rate	Up to 15 Hz
Field-of-View	45°
Max Range	100 m
Beam Width	1° x 20°
Number of Beams	256
Beam Spacing	0.18°
Range Resolution	1.0 in.

A. Forward-Looking Sonar Development

The first step in developing the ROV simulation was building the model for the forward-looking sonar. Fortunately, the SICK laser model, shown in Figure 10(a), in MORSE provided a starting point. The SICK model uses ray tracing from the origin of the model to a detected obstacle in order to build an array of laser points. The SICK model was modified to allow for ray tracing to not only occur in the X and Y directions, but also in the Z-direction in order to model the sonar's beam characteristics. The result after adding beams in the Z-direction is shown in Figure 10(b).



(a) SICK laser model. (b) Forward-looking sonar model.

Fig. 10: The SICK and forward-looking sonar MORSE models.

In order to publish the sonar data from MORSE to the ROS framework, a new ROS sensor message had to be created. This was accomplished by creating the `auv_msgs` ROS package with the `sonar_cloud` message defined by:

Listing 1: Sonar Cloud Definition

```
uint8 NumPlanes uint8 NumPointsPerPlane
float32 angle_min float32 angle_max
float32 angle_increment float32
range_min float32 range_max float32
vert_height float32 vert_resolution
float32[] x float32[] y float32[] z
```

The Python script that is executed by the sonar model in MORSE generates the `auv_msgs::sonar_cloud` message, which is then converted to a true Point Cloud by the custom `sonar_point_cloud` node. The `sonar_point_cloud` node uses the `sonar_cloud` message's `NumPlanes` and `NumPointsPerPlane` variables to unravel the one-dimensional array of rays into the three-dimensional point cloud. The `sonar_point_cloud` node is part of the `auv_morse` package developed during this research to manage all sonar image creation, sonar image processing, obstacle detection, and ROV motion model nodes. The Point Cloud Library (PCL) was developed by Willow Garage specifically for ROS nodes. An overview of the sonar data flow from MORSE to ROS is shown in Figure 11. The `sonar_point_cloud` node also publishes an OpenCV image, which is subscribed to by the `sonar_process` node. This node uses the OpenCV library to detect corners and edges that are translated into obstacles and features for obstacle detection.

The creation of the sonar image takes places in several steps. First, the `auv_msgs::sonar_cloud` message is converted into a 3D point cloud. Then the Z-component from each point in the point cloud is removed and the point is drawn in a 2D OpenCV matrix, which is essentially a projection of the 3D point cloud into 2D space. The image is scaled up in size by multiplying all of the X and Y components of the rays in the image by a times ten scaling factor. This helps to separate features from each other and produces an image that is closer to the image output from a real sonar. In order to simulate basic sonar noise, a Gaussian blur algorithm is convoluted across

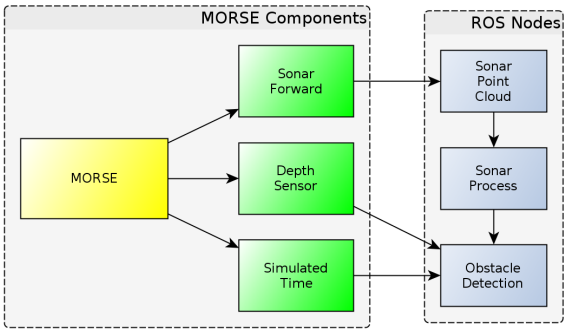


Fig. 11: MORSE and ROS node interactions.

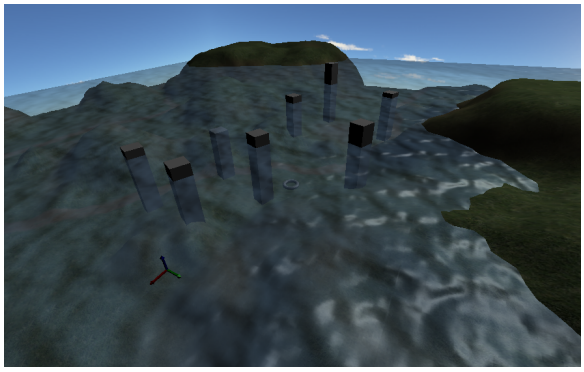


Fig. 12: The underwater MORSE environment.

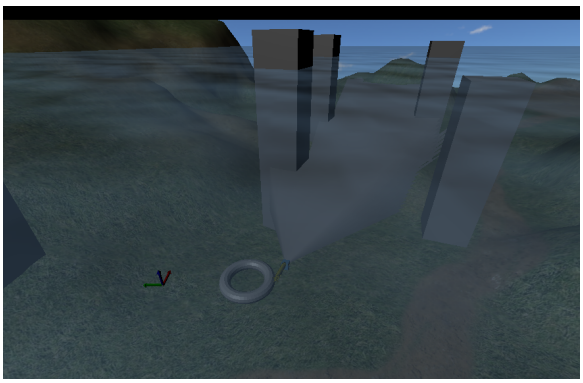


Fig. 13: Forward-looking sonar detecting obstacles.

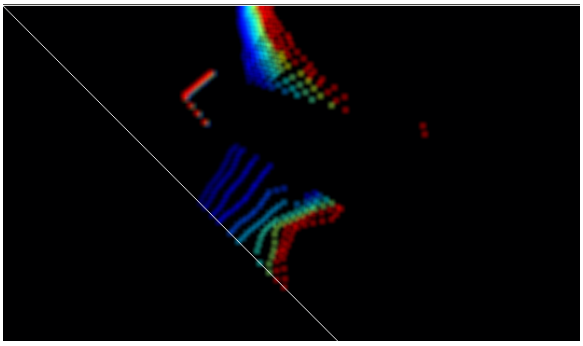


Fig. 14: Screen capture of forward-looking sonar image.

the image. Finally, the white lines designating the sonar image boundaries are drawn after the Gaussian blur so that they are not affected by the noise.

B. Sonar Model Simulation Results

As shown in Figure 12, several rectangles were added to the environment to simulate pier support beams. A torus was also added to the environment to provide an interesting object on the sea floor for the ROV's sonar to detect. After the simulation was initialized and executed, the ROV began to move forward at a constant velocity, while using its forward-looking sonar to avoid nearby obstacles. The external ROS nodes were processing the published sonar images and detecting features in real-time. The ROS nodes were able to process the sonar images faster than MORSE could generate the sonar images due to the computational complexity of ray tracing.

The forward-looking sonar on the ROV used Blender's built-in ray tracing functions to detect obstacles in the environment upon simulation execution. The ray tracing out of the sonar sensor can be seen in Figure 13, where the sonar rays are colliding with the pier support beams and some of the surrounding bathymetry. An example of the sonar image output from the `sonar_point_cloud` and MORSE ray tracing simulator is shown in Figure 14. It should be noted that this sonar image is rotated compared to standard sonar images. The choice to put the apex of the sonar image at the top-left of the image was due to the fact that the OpenCV drawing functions use a coordinate system that place the origin at the top-left of the matrix or image. A final note about Figure 14 is the sharp red corner that is seen closest to the sonar image's apex. This red corner is a rectangular pier support beam and an excellent candidate for a feature that could be fed to a navigation algorithm. The brightly colored areas to the sides of the beam are the detected sea floor, but they do not provide strong features for a navigation algorithm. It is important to note the acoustic shadow behind the pier support beam because the pier beam extends all the way to the water's surface.

In other papers, such as [6], [28], and [5], the authors demonstrated the sensing of known 3D objects with their sonar simulators. Thus, a torus was placed in the environment, so that the sonar image of the torus could be examined. The 3D model of the torus is shown in Figure 15 and the sonar image output from sensing the torus is shown in Figure 16. The sonar image of the torus is interesting because the human eye can almost detect a 3D object in the 2D image. The acoustic shadows that are created behind the inner circle of the torus and behind the torus give the image depth.

VI. Human Diver Model

A human diver model is required for any type of UHRI scenario. The requirements for a basic human diver model consist of a realistic 3D model that can be imaged by the VideoRay's simulated sonar sensor and the simulated optical camera. Also, the diver's motion model must capture the diver's ability to rotate in place, move in the forward direction, and move vertically within the water column.

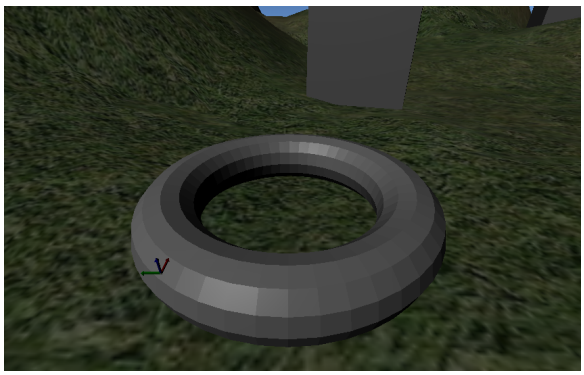


Fig. 15: 3D model of torus.

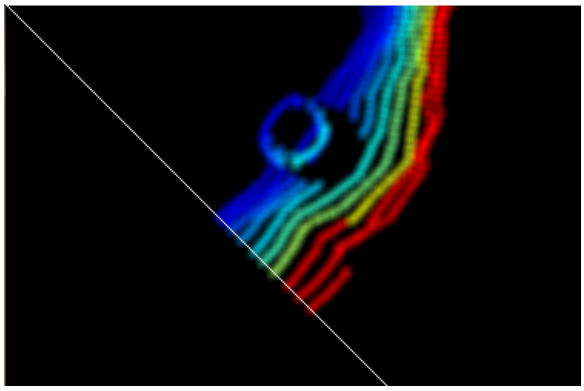


Fig. 16: Image of torus structure via 2D imaging sonar.

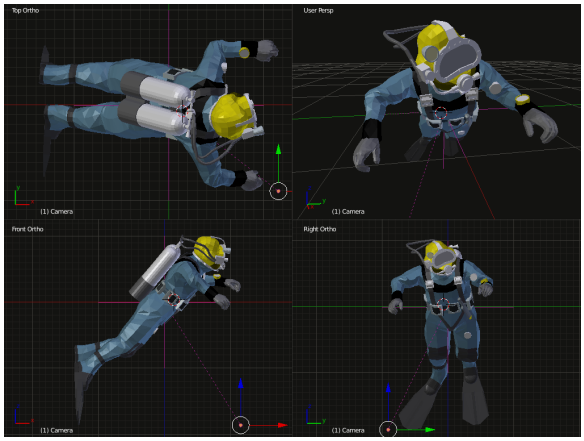


Fig. 17: Quad view of the Diver 3D model.

A. Human Diver 3D Model

The diver model was acquired through the Google Sketchup 3D Warehouse and exported from Google Sketchup using the VRML format. Once the model was imported into Blender, the extraneous meshes, such as the diver's air bubbles were removed from the model. The final quad view human diver model is shown in Figure 17.

B. Human Diver Dynamics Model

The motion model for the human diver is fairly simple. In order to extend air supply, a diver will move in smooth, constant velocity trajectories. Since a diver can essentially rotate in place, the diver's motion will also assumed to be holonomic. Given the motion model's description, the diver's motion model can be described on the X-Y plane by the motion model of a differential drive robot. The system equations for a differential drive robot are defined by:

$$\dot{x} = \frac{r}{2}(u_l + u_r)cos(\theta) \tag{10}$$

$$\dot{y} = \frac{r}{2}(u_l + u_r)sin(\theta) \tag{11}$$

$$\dot{\theta} = \frac{r}{L}(u_r - u_l) \tag{12}$$

where r is the radius of the wheel, L is the distance between the two differential wheels, x is the x-position, y is the y-position, θ is the heading, u_l is the angular velocity of the left wheel, and u_r is the angular velocity of the right wheel. The differential drive motion model was adapted to the diver model by assigning the following values: $L = 0.5$ and $r = 1$. Heading control for the diver model was accomplished by proportionally driving the u_l and u_r inputs. Similar to the VideoRay model, the motion in the Z-direction is decoupled from the motion in the X-Y plane and the velocity in the Z-direction is provided by:

$$\dot{z} = u_z \tag{13}$$

where u_z is the vertical control input. The diver is assumed to be at neutral buoyancy, which is an adequate assumption for a diver that is not wearing weighted boots. Divers that swim vertically through the water column will wear additional weights until they are at neutral buoyancy because it reduces the amount of effort required to remain at a constant depth while working. Future work will result in the development of a diver model for a commercial diver wearing weighted boots.

VII. Mission and Simulation Configuration

While open-source publish-and-subscribe systems, such as ROS, have been heavily used within the robotics and autonomy community for field robotics work, ROS does not lay the framework for an autonomy *architecture*. An autonomy architecture that allows for tracking contacts, defining state machines, fusing autonomous behaviors, setting mission parameters, and controlling and monitoring the robot during mission deployment is required to provide an autonomous system with the best chance of completing its mission. The Mission Oriented Operating Suite (MOOS) coupled with IvP provides such a framework. MOOS is a publish-and-subscribe middleware layer for robotic platforms. IvPHelm is the MOOS module that provides the autonomy framework for maritime systems. To demonstrate the use of MOOS-IvP with MORSE, an example mission was created that consisted of the VideoRay performing track-and-trail of the diver.

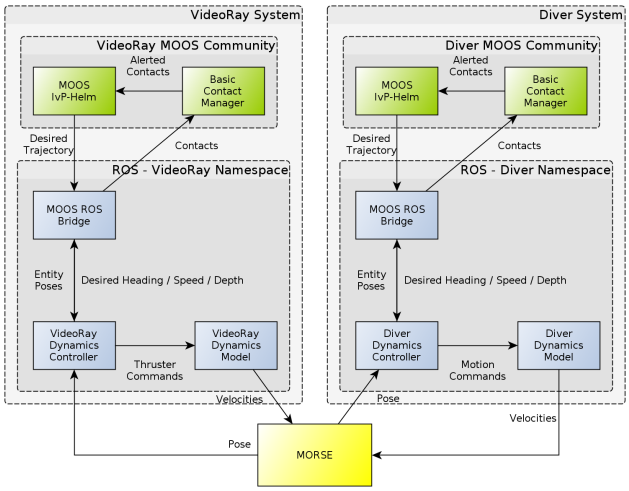


Fig. 18: The diver and VideoRay systems individually communicating with the MORSE simulator.

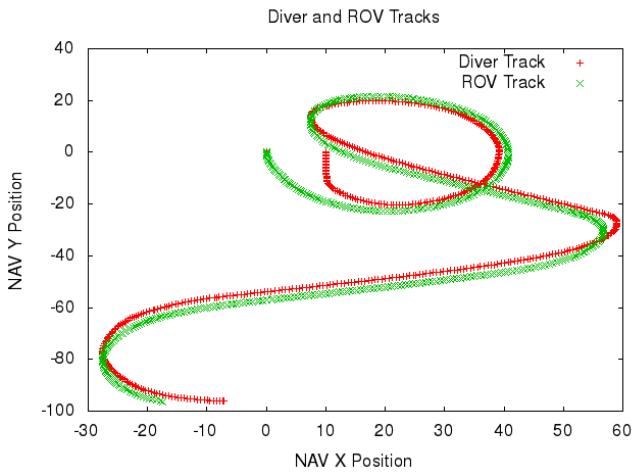


Fig. 19: Diver and VideoRay tracks during track-and-trail mission.

A. *IvPHelm Behavior Configuration*

The IvPHelm process within MOOS is the behavior fusion engine. The main advantage of a behavior fusion engine, versus a simple station machine, is that it is able to resolve conflicting goals in an uncertain and changing World Model. For example, a behavior fusion engine can be used to avoid obstacles while transiting to a goal location by resolving the conflicting goals of the transit behavior and the avoid obstacle behavior. In this scenario, the diver's IvPHelm configuration is fairly simple. It consists of the `BHV_ConstantDepth` and `BHV_Loiter` behaviors. In this case, the diver's `BHV_AvoidCollision` is not enabled because it is assumed that the diver confidently assumes that the ROV will avoid the diver. The VideoRay ROV's IvPHelm configuration is slightly more complicated. The ROV is also executing the `BHV_ConstantDepth` behavior, but the behavior is receiving diver contact reports from MORSE that update the desired depth. In order to avoid collisions with the diver, the ROV's `BHV_AvoidCollision` behavior is enabled, which is updated based on diver contact reports. Finally, the ROV's `BHV_Trail` behavior allows the ROV to follow the diver. The trail behavior is configured to follow the diver at a 180° angle with a trail range of 2 m.

B. *Mission Launch Configuration*

The overall mission consists of three main components: the MORSE simulator, the diver system, and the VideoRay system. Both the diver and VideoRay systems consist of similar sub-components that could conflict with each other, if not configured correctly. Fortunately, the ROS system provides namespaces to separate nodes of similar names and the MOOS framework provides network separation through MOOS communities and MOOS Bridges that can shuttle data between separate communities. Thus, through proper configuration, a single launch file was used to launch both the VideoRay and diver systems at the same time, both of which individually communicated with the MORSE simulator, as depicted in Figure 18.

VIII. Mission Execution and Results

Upon execution of the mission, the diver begin to perform loiter patterns based on the coordinates defined in the IvPHelm behavior file. Also, the VideoRay begins to transit towards the diver based on contact reports it received from MORSE. In this mission example, the diver contact report was generated from truth simulated data from MORSE instead of the processing of the simulated sonar data to test the behavioral algorithms, not the machine vision algorithms. Based on previous work [3], it is assumed that the diver's position can be detected and tracked from forward-looking sonar data. The simulated X-Y trajectories of the diver and VideoRay are plotted in Figure 19. From the trajectories, it can be seen that the ROV transits towards the diver and then proceeds to track-and-trail the diver throughout the duration of the mission.

A third MOOS community was also configured to act as a base station for the VideoRay and diver. This MOOS community consisted of the pMarineViewer application and several MOOS bridges to gather position information from the diver and VideoRay. During mission execution, the positions and headings of the diver and VideoRay were displayed in pMarineViewer. An instance of the VideoRay trailing the diver in pMarineViewer is shown in Figure 20. Finally, a 3D screen capture of the track-and-trail mission in progress is shown in Figure 21.

IX. Discussion

The track-and-trail demonstration between the ROV and the diver was successful in testing the flow of data between MOOS, ROS, and MORSE. The obstacle avoidance and contact management, which were required in the track-and-trail example, could be used for various other and more complicated examples. For example, with just the trail and obstacle avoidance behaviors, vessels can move in formation or navigate busy shipping lanes.

The use of MORSE as the main simulation engine has some major benefits that are not necessarily obvious at first.

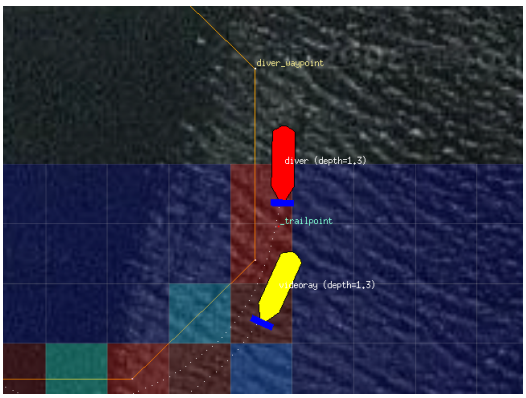


Fig. 20: Visualization of ROV trailing diver in pMarineViewer. The diver is represented by the red UUV symbol and the VideoRay is represented by the yellow UUV symbol.

Fig. 21: The VideoRay trailing the diver in the MORSE simulator.

Not exploited in this demonstration, robots and human avatars within MORSE can pick up and move objects within the scene. Also, since MORSE is built upon the Blender Game Engine, events can be triggered when certain conditions within the environment are satisfied. The fact that Blender is capable of importing, manipulating, and building complete 3D models means that with just basic Blender experience, any user can quickly and easily add new robots, scenes, and objects to the MORSE simulation environment.

Another interesting aspect of this simulation environment is that the programmer can choose to either use Blender's internal integration engine for basic physics models, such as the differential drive robot, or the programmer can integrate complex models with the use of an external C++ program, which was done for the VideoRay model. This allows for adjusting the fidelity of the simulation without major changes to the simulation environment. While most of the physics models in MORSE utilize the internal Blender physics engine, it will be useful for the community to develop a standard messaging scheme when using the external physics models.

X. Conclusions and Future Work

While the development of the UHRI simulator is still in progress, the system has proven to be successful for testing basic UHRI scenarios. With an active developer community, MORSE is currently under heavy development. As MORSE improves and becomes easier to use, so will the UHRI simulator.

The UHRI simulator requires further development before it can fully simulate human-robot interactions. First, the diver model must be adapted to allow the diver to grasp and "use" tools. Likewise, the ROV must be able to use tools, which will result in some sort of change in the environment. This would allow the simulation of a human-robot team for underwater construction or salvage. Second, controlling the human via script files would decrease development time. Currently, the human is controlled through IvPHelm, which is specialized for controlling the movement of ship vessels. It is important that the VideoRay is controlled by a full autonomy architecture because it has to be able to adapt to a changing environment, but the diver is a human that could be scripted. Finally, either IvPHelm will have to be augmented or a new autonomy architecture focused on human-robot interaction will have to be developed. Since IvPHelm is specialized for vessel movements, it does not have behaviors for grasping tools, using tools, or order-based planning. An autonomy architecture for human-robot interactions will require a different set of input behaviors than are currently provided by IvPHelm.

Acknowledgment

The authors would like to thank the Georgia Tech Research Institute (GTRI) for supporting this work. Also, the authors would like to thank VideoRay for providing the 3D model for the VideoRay Pro 4.

References

[1] D. G. Gallagher, "Diver-based rapid response capability for maritime-port security operations," in *OCEANS 2011*, 2011, p. 110.

[2] W. Bluethmann, R. Ambrose, M. Diftler, S. Askew, E. Huber, M. Goza, F. Rehnmark, C. Lovchik, and D. Magruder, "Robonaut: A robot designed to work with humans in space," *Autonomous Robots*, vol. 14, no. 2-3, pp. 179–197, 2003.

[3] K. DeMarco, M. E. West, and A. M. Howard, "Sonar-based detection and tracking of a diver for underwater human-robot interaction scenarios," in *SMC 2013*. IEEE, Oct. 2013, pp. 1–8, (Accepted July 2013).

[4] J. M. Bell and L. M. Linnett, "Simulation and analysis of synthetic sidescan sonar images," in *Radar, Sonar and Navigation, IEE Proceedings-*, vol. 144, 1997, pp. 219–226.

[5] E. Coiras and J. Groen, "Simulation and 3D reconstruction of side-looking sonar images," *Advances in Sonar Technology*, p. 114, 2009.

[6] D. Gueriot, C. Sintes, and B. Solaiman, "Sonar data simulation & performance assessment through tube ray tracing."

[7] D. Guriot and C. Sintes, "Forward looking sonar data simulation through tube tracing," in *OCEANS 2010 IEEE-Sydney*, 2010, p. 16.

[8] J. Sattar and G. Dudek, "Where is your dive buddy: tracking humans underwater using spatio-temporal features," in *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, 2007, pp. 3654–3659.

[9] ——, "A vision-based control and interaction framework for a legged underwater robot," in *Computer and Robot Vision, 2009. CRV'09. Canadian Conference on*, 2009, pp. 329–336.

[10] H. Buelow and A. Birk, "Gesture-recognition as basis for a human robot interface (HRI) on a AUV," in *OCEANS 2011*, 2011, pp. 1–9.

[11] B. Gerkey, R. T. Vaughan, and A. Howard, "The player/stage project: Tools for multi-robot and distributed sensor systems," in *Proceedings of the 11th international conference on advanced robotics*, vol. 1, 2003, pp. 317–323.

[12] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, vol. 3, 2004, pp. 2149–2154.

[13] "Microsoft robotics developer studio." [Online]. Available: http://www.microsoft.com/robotics/

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

[15] E. Coumans *et al.*, "Bullet physics library," *Open source: bulletphysics. org*, 2006.

[16] R. Smith, *Open dynamics engine (ODE)*, 2006.

[17] R. Osfield, D. Burns *et al.*, *Open scene graph*, 2004.

[18] G. Echeverria, N. Lassabe, A. Degroote, and S. Lemaignan, "Modular open robots simulation engine: Morse," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, 2011, pp. 46–51.

[19] G. Metta, P. Fitzpatrick, and L. Natale, "YARP: yet another robot platform," *International Journal on Advanced Robotics Systems*, vol. 3, no. 1, pp. 43–48, 2006.

[20] "pocolibs - openrobots." [Online]. Available: http://www.openrobots.org/wiki/pocolibs

[21] K. DeMarco, M. E. West, and T. R. Collins, "An implementation of ROS on the yellowfin autonomous underwater vehicle (AUV)," in *OCEANS 2011*, 2011, p. 17.

[22] T. I. Fossen, *Guidance and control of ocean vehicles*. West Sussex, England: John Wiley & Sons, 1994.

[23] W. Wang and C. M. Clark, "Modeling and simulation of the VideoRay pro III underwater vehicle," in *OCEANS 2006-Asia Pacific*, 2007, p. 17.

[24] P. M. Newman, "MOOS-mission orientated operating suite," *Massachusetts Institute of Technology, Tech. Rep*, vol. 2299, no. 08, 2008.

[25] "Robot operating system (ROS)." [Online]. Available: http://www.ros.org/wiki/

[26] K. Ahnert and M. Mulansky, "Odeint-solving ordinary differential equations in c++," *arXiv preprint arXiv:1110.3397*, 2011.

[27] "High resolution MultiBeam imaging sonar - BlueView technologies." [Online]. Available: http://www.blueview.com/products.html

[28] D. Gueriot, C. Sintes, and R. Garello, "Sonar data simulation based on tube tracing," in *OCEANS 2007-Europe*, 2007, p. 16.