Unity Underwater ROV Simulator

Kun Seng Vu
Macau Anglican College, Macao SAR
(rickyvu189@gmail.com)

***Abstract*— The purpose of this paper is to describe an array of tools for simulating underwater vehicles in the virtual reality software platform Unity. The advantage of using Unity is that it provides a helpful interface to modify an object's attributes and its scripts easily. This allows the user to change and adapt the simulator to different environmental situations.**

The concept consists of using functionality scripts for control and add-on scripts to compute water physics. The add-on scripts can be used to test ideas that extend the capability of the pre-existing rigid ROV body system. There are two basic scripts needed; one for buoyancy and another for drag. Unity's rigid body system does contain a primitive option for adding drag to an object. However, it is a rough estimate only providing a force that counteracts the velocity of a moving object at its center of mass. This would be acceptable for a simple 3d shape but not for the complex shape of an ROV. Therefore, a more intricate drag script algorithm is required.

The main purpose of developing accurate simulation scripts that are true to nature is to keep the simulation close to reality. The main constraint that needs to be considered is to maintain computational complexity at a reasonable level that allows the average computer to run the simulation at 60fps.

The end result is an easily accessible simulated environment for testing the system software and the design of ROV and AUV. A model of the vehicle can be added into the simulation. By using Unity's sockets feature, camera footage of the simulated environment can be sent to the main system software where calculations can be done. The results of the calculations can be sent back to the simulation to adjust the movement of the vehicle. By adding simple models for props, it is possible to also test computer-vision related tasks with this simulation.

This paper specifically details the simulations used to test the 2019 Macau Anglican College ROV in the Unity Software in preparation for competitions.

Keywords: ROV, AUV, Simulation, Unity Software, Unity Scripts, C#

I. INTRODUCTION

Simulation software is software that models the design and/or performance of products under different conditions [1]. It is the imitation of the performance of an object experiencing known conditions over time [2]. It is extensively used in the fields of engineering, physics, chemical science, biology and medicine [1]. Simulation modeling helps engineers and designers to understand the performance of applied designs without actually performing any physical operation [3]. Given that the testing environment can also be part of the simulation, there can be considerable savings in costs, manpower and time.

It is important to mention that a simulation is usually based upon a set of mathematical equations that are representative of the equipment being observed, the surroundings and the environmental conditions the equipment is being tested for. It is only an imitation [4]. It does allows repeatability a very important aspect of science and in this case the analysis and further collection of data.

Simulation software has been used to design and test equipment like buildings, bridges, cars, aircraft and space vehicles. It has also been used to assess the performance of equipment and materials in specific situations.

The main purpose is for us to test the system software of our ROV and potentially a future AUV. Since this simulator is accessed through Sockets, it is a completely isolated system capable of running on its own. This means we can connect any system programs to it as long it has the appropriate socket connection. This is the most important aspect as this means we can easily reuse the same simulator to assess the system software of other ROVs/AUVs.

It can also be used to do a variety of different tests. The setup of the underwater vehicle is a major concern. We can know if the thruster setup and directions will affect the movement stability. By assembling the vehicle's thrusters, ballasts and floats in simulator, it is possible to identify the best combination between mobility and stability for our design.

Another thing we can find out is the speed the ROV/AUV can achieve. By experimenting with a real thruster, gaining data and input it into the simulator, we can have a general idea of the maneuverability of the underwater vehicle. This is especially important as both fine precision control and fast mobility are crucial in a task completion competition setting.

As mentioned before, the repeatability of a simulator can be used to generate large amounts of useful data. This means we can create many scenarios to test out our computer vision tasks. For example by changing the shades of color and/or adding slight shadows to test the robustness of our computer vision programs.

Another useful aspect of the simulator is that it can potentially be a tool for pilot training. Since every in-water test is time consuming and halts most of the team development/production time. By completely isolating pilot training from water/tool test can improve team efficiency.

II. GENERAL PROGRAM STRUCTURE

The code was written completely in Microsoft's C# programming language, version 8. There were a total of 6 objects set up in the Unity software for the simulation. 3 were for the ROV and 3 were for the environment. Note that two environmental objects were set up even though their use would depend upon the simulation being carried out. These have been listed below:

i. Objects in the simulator

1. ROV
   - Main body
   - Thrusters
   - Camera
2. Environmental
   - Floor
   - Pool side walls (Optional)
   - Props for testing the vehicle (Optional use)

All of these objects are quite obvious and were set up and configured in the software. This all forms the software versions of the ROV hardware and the environment.

A total of 6 scripts were then written to act or influence the objects that had been set up as listed above in section A [5]. These had to do with the movement of the ROV; Movement and Thrusters. Scripts were also written for the forces that could influence the ROV movement; Buoyancy and Drag. Two scripts were dedicated for the software use; Camera recording and a Socket Server. The latter is an interface into the simulation software for data flow in either direction. A detailed description for each software script is included below.

ii. Software Scripts

- Movement
- Socket server
- Thrusters
- Camera recording
- Buoyancy force
- Drag force

a. Movement

This script is used to move the simulated vehicle directly through simple key bindings allowing the simulated ROV to be controlled directly from the computer keyboard. This was added for testing the simulator without needing to connect to the entire ROV/AUV system. Hence, this script was designed to be as simple as possible.

b. Socket

The simulator program needs a way to connect with the ROV/AUV system. This is done through the use of socket I/O [6]. The simulator and control program can act as clients to connect through an external socket server.

The socket feature offers many possibilities. It provided a way to test our control programs. The control program is used to send commands to the simulator to move the simulated vehicle and to interact with the simulated environment. A simulated camera recording can also be sent through the socket. Programs can then decompress and process the data just like camera feeds from real life cameras. This enables the testing of computer vision programs.

c. Thruster

This script was designed so that each thruster could be assigned an address. The control program then only has to send a command that consists of a target address and power level to operate a thruster. All thrusters receive these commands and the thruster object code checks the received address data for a match. If an address match is confirmed, then the thruster uses the power level to set the thruster speed.

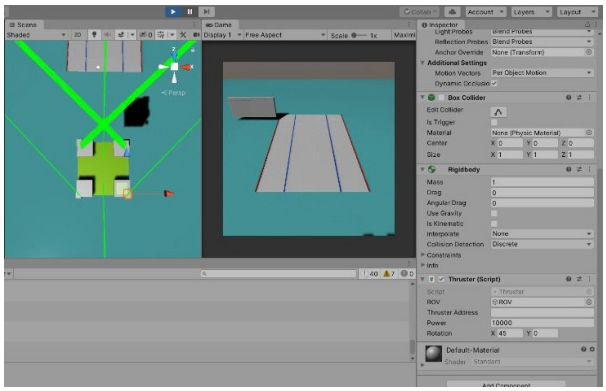

Figure 1. A screenshot showing 45 degree thruster angle on two thrusters

Physically in the simulation, the thruster will apply the force towards the direction it is facing. So each thruster pointing direction is angled manually when assembling the virtual vehicle. This is an important consideration since real, non-simulated, ROVs have thrusters aimed at slight angles for stability and to ensure the ROV will travel along a straight trajectory. Figure 1 shows a screen shot of a simulated vehicle with a thruster angle of 45 degrees on the two rear thrusters.

d. Camera Recording

The camera recording script is fundamental for testing computer vision tasks. This allows for re-runs of what the simulated ROV "sees" as it moves about in the simulated environment.

Most of the code is standard, used in Unity in-game cameras. [7] However, after acquiring the photo, we will encode it and store it as a variable. It is compressed to PNG format to improve data transfer when the image is transported through sockets to the ROV/AUV system.

e. Buoyancy

The buoyancy of the simulated ROV had to be calculated and the value depends upon several factors.

Usual formula in physics

Traditionally the calculation uses a simple formula as shown below.

$$F_b = \rho g V$$

Where:

F_b - Buoyancy force that is acting on the object
V - Submerged volume of the object
ρ - Density of the fluid the object is submerged in
g - Acceleration due to gravity

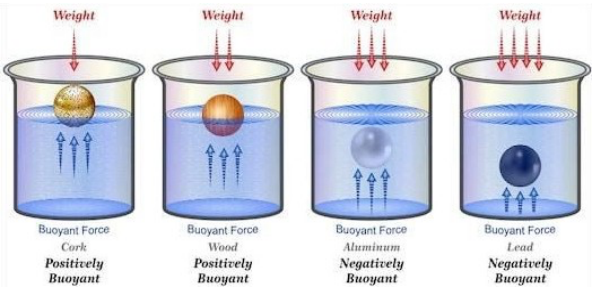

Figure 2. Demonstrating how buoyancy affects an object in water

However, in this case, Density and gravitational acceleration are preset values from the user. This means we only need to find the volume of the object to acquire the buoyancy.

There are two options here. One is to measure or calculate the volume of the object in real life. The other method is to calculate mass over density for the object.

$$V = m/\rho_o$$

Where:
- V - Submerged volume of the object
- M - Mass of the object
- ρ_o - Density of the object

The first method is relatively simple at first glance, especially for the cases when the single object may be composed of a couple different materials. However, the drawback is that it is difficult to measure the volume for objects with irregular geometry. Measuring by water displacement is possible, but not an ideal solution.

Alternative formula

An alternative method of calculating buoyancy exists that requires the density and mass of an object. Mass of the ROV can be measured using an electronic balance. Density can be searched up online. An example would be this: [8]

$$F_b = gm\rho_f/\rho_o ,$$

Where:
- F_b - Buoyancy force that is acting on the object
- ρ_f - Density of the fluid the object is submerged in
- ρ_o - Density of the object
- m - Mass of the object
- g - Acceleration due to gravity

We can use this formula to calculate the buoyancy force. Then multiply this force to a vector pointing upwards. Then add this force to the object constantly.

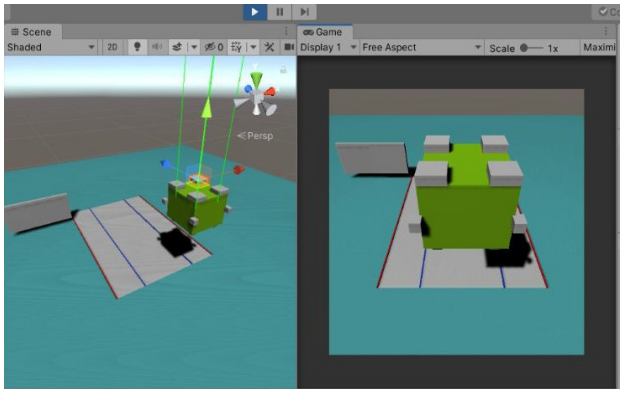

Figure 3. A ROV working with buoyancy. The 4 gray boxes on top of the ROV are the floats. Green lines represent the buoyant force of the float.

f. Drag

Usual formula in physics

$$\text{Drag force} = \tfrac{1}{2} C\rho A_p (v^2)$$

Where:
- C - Drag coefficient
- A_p - Area of the object facing the fluid
- ρ - Density of the fluid
- v - Velocity of the object

In this case, we can simplify this slightly since the density of fluid remains the same. All the constants can be grouped together as one.

Modified formula

$$\text{Drag force} = CAp(v^2)$$

Where:
- C - Drag coefficient (which includes ½ and ρ)
- Ap - Area of the object facing the fluid
- v - Velocity of the object

This simplifies the tweaking process since only one value needs to be changed instead of two. The calculation for drag also requires the surface area of the object. However, Unity doesn't provide this information to users, therefore, an indirect method must be used. This method involves using meshes.

Explanation of Meshes

A mesh consists of triangles arranged in 3D space to create the impression of a solid object. Each triangle is defined by its three corner points or vertices.

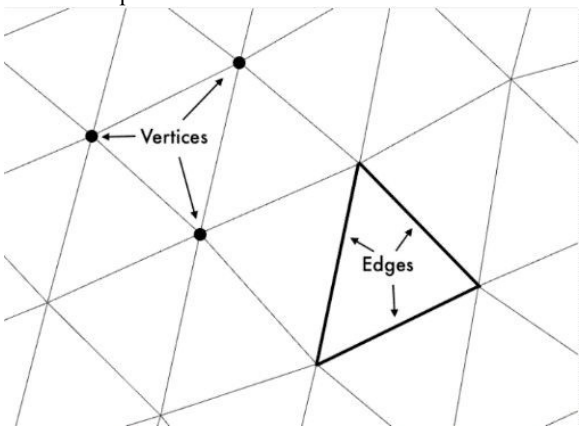

Figure 4. A visual mesh showing the position of the vertices and edges.

Mesh Calculations

By using the vertices of each mesh, we can calculate the center point of each individual mesh triangle. Suppose we have 3 coordinates, each one for one corner of the mesh triangle. We can call them V1, V2 and V3.

$$(x_1, y_1, z_1) \ldots V_1$$
$$(x_2, y_2, z_2) \ldots V_2$$
$$(x_3, y_3, z_3) \ldots V_3$$

The center point will then be adding up their components individually and dividing by 3.

$$\left(\frac{x1+x2+x3}{3}, \frac{y1+y2+y3}{3}, \frac{z1+z2+z3}{3}\right)$$

Vector for the edges can be found by subtracting V2 from V1 and V3 from V1

$$V_A = V_1 - V_2 = (x_1-x_2, y_1-y_2, z_1-z_2)$$
$$V_B = V_1 - V_3 = (x_1-x_3, y_1-y_3, z_1-z_3)$$

By using the cross product on the 2 edge vectors (a & b), the result is another vector, a normal vector, that is orthogonal to vectors a and b. The magnitude of this normal vector is the area of a parallelogram formed by vectors a and b. Dividing this by 2 will give us the area of the mesh triangle. See the formula below:

$$V_N = \frac{V_A \times V_B}{2}$$

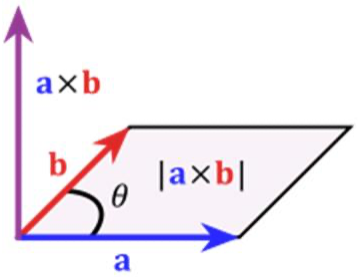

Figure 5. Visual representation of cross products between two vectors

The Unity software has some inbuilt functions that can be used to calculate several parameters from mesh vectors. From these functions it is possible to determine the direction the mesh triangle is moving in.

Projected area can then be found by evaluating the dot product between Velocity and cross product of the two edges. This can be represented by the following formula. The visual representation is in Figure 6.

$$AP = V \cdot V_N$$

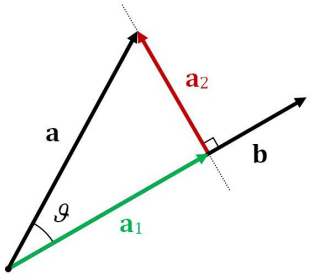

Figure 6. Visual representation of dot product between vectors a &b Dot product is the length of a_1 multiplied by the length of b (scalar)

The drag coefficient (C) comes from user input. So this means we have all the required data for calculation of drag.

$$F_D = CA_P |V|^2$$

Drag is merely a value; we still have to apply it as a force onto the correct position. Since drag is the force against velocity this force will be in the opposite direction as velocity.

Last step, because we measured velocity at the center of each mesh triangle, the drag force is applied to its respective mesh center. This last step makes the model complete.

$$V_D = F_D \times \frac{V}{|V|}$$

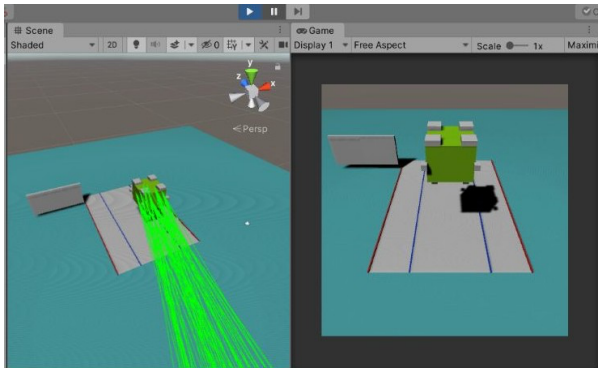

Fig. 7 A simple ROV shown with drag. Green lines represents the force of drag

III. Limitations

There are some drawbacks involved with this method. Drag in real life applies to every point on the surface of an object. This algorithm only applies drag to the center of the mesh triangles that make up the object. The accuracy will heavily depend on the number of triangles forming the object. In theory, 3D models with more triangles should give a more accurate result. But this will also requires more computational power. Therefore, shapes such as cylinders are made into hexagons and small details are removed for a smoother result in terms of viewing the simulation.

Another issue is with the simulation of the drag. This program assumes that the object being tested is totally exposed and not protected from any current producing drag. This applies to any objects inside the object (ROV). This means that 3D models cannot have much space inside them to help reduce the exaggerated drag force calculated.

IV. Future Improvements

Currently, the simulator considers the whole environment as underwater. One refinement would be to define an area as water and only apply the buoyancy calculation to the sections of the ROV that are actually underwater. This would help simulate the performance of the ROV at the surface, allowing for more accurate simulations to be performed.

Another enhancement is to better utilize the Graphics Processing Unit (GPU) for better performance. Currently this simulation, using basic, relatively simple geometry can barely reach the viewing update speed of 60 fps. The CPU may not be an efficient enough method for simulating complex geometry. Better use of the GPU may solve this problem.

V. Conclusions

This simulator is a useful tool for testing the movement of an ROV. Moreover, it is an essential tool for the testing of computer vision programs. The versatility and repeatability of this simulator is the main advantage. It can greatly assist the development of ROV and AUV.

The programming is in C#, and uses a simulation / game platform, Unity, to model gain information and experience on something that is essentially engineering.

It is an example of the application of vectors, dot product and cross products as well as a direct theoretical application of the formulas of forces, buoyancy and drag. This project allows users to see how close simulations can be to the real thing.

VI. Acknowledgements

This project would not have been possible without the generous support of the Macau Anglican College for providing me with resources and workspace. The FDTC were generous with the provision of funding for this project. My Mentor, Mr Andy Tsui, for giving me support and guidance. Finally, the IEEE for the chance to attend the HK IEEE CE/OES YE20 Conference. Thank-you all.

VII. REFERENCES

[1] Pfluger, D., Valintin, J., Mehl, M., Lindner, F., Pfander, D., Wagner, S., Graziotin, D., Wang, Y., (2016). The Scalability-Efficiency/Maintainability-Portability Trade-Off in Simulation Software Engineering: Examples and a Preliminary Systematic Literature Review Retrieved from: https://www.researchgate.net/publication/318224822. Accessed: 1st Dec. 2020.

[2] Shannon, R.E., 1975. *Systems Simulation – The Art and Science*, Prentice-Hall.

[3] Banks, Carson, Nelson & Nicol. "Introduction to Simulation". Retrieved from: https://cs.wmich.edu/alfuqaha/Spring10/cs6910/lectures/Chapter1.pdf Accessed 28th Nov. 2020.

[4] Ingalls R. G., (2008) Introduction to Simulation. *Proceedings of the 2008 Winter Simulation Conference S. J. Mason, R. R. Hill, L. Mönch, O. Rose, T. Jefferson, J. W. Fowler eds.*

[5] Technologies, Unity. "Creating and Using Scripts." Unity. Retrieved from: https://docs.unity3d.com/Manual/CreatingAndUsingScripts.html. Accessed 1st Feb. 2021.

[6] Karelz. "System.Net.Sockets Namespace." Microsoft Docs. Retrieved from: https://docs.microsoft.com/en-us/dotnet/api/system.net.sockets?view=net-5.0. Accessed 1st Feb. 2021.

[7] Technologies, Unity. "Camera." Unity. Retrieved from: https://docs.unity3d.com/Manual/class-Camera.html. Accessed 1st Feb. 2021.

[8] "Densities of Materials." Engineering ToolBox..Retrieved from: https://www.engineeringtoolbox.com/density-materials-d_1652.html. Accessed 1st Feb. 2021.

[9] Technologies, Unity. "Mesh." Unity. Retrieved from: https://docs.unity3d.com/ScriptReference/Mesh.html. Accessed 1st Feb. 2021.

View publication stats