Noetic Underwater Simulator (NUSim)

AKASH CHAUDHARY

Compiled July 24, 2020

This document serves as a Preliminary Design Report, and will highlight the tasks undertaken and steps followed in completing the first step towards building a complete Simulation for an Unmanned Underwater Vehicle (UUV) using Unity3D and Robot Operating System (ROS). In this version, we are going the build the environment in Unity3D and test it using keyboard controls without ROS integration.

1. INTRODUCTION

Simulations are a crucial part of the Design and Deployment process as it gives us an opportunity to test our software and algorithms before deploying them in the physical world. This not only ensures that the operators are well versed with the system that they are working with but also reduces the risk of deploying the product directly which can cost the user their hardware in case of failure. In this project, we are trying to build an Open Source Simulator to test the control system for Unmanned Underwater Vehicles (UUV). We will be using Unity3D, a physics engine to simulate the environment in which the UUV will be deployed, and Robot operating System (ROS) to build the control system for the same. This proposal deals with the first step towards that goal and aims to build the 3D environment in Unity3D with keyboard controls and no ROS Integration. This proposal highlights the steps and goals of the first version. These might change as we progress through the project, but this proposal can serve as a guiding base for the development process.

The Unity3D project will be uploaded to GitHub so that it can be accessed easily. The updated GitHub repository can be found here.

2. MISSION STATEMENT

The mission statement will highlight the tasks to be undertaken by the UUV after deployment from a remote Control Station.

1. Start from surface near the Control station.
2. Sink down
3. Find and Follow the Pipeline.
4. Search for the leak in the pipeline.
5. Cross gate check points on the way back.

3. THE SIMULATION

The simulation starts with the AUV attached to the control station ready for the launch. As soon as the user presses 'R', the AUV is released from the station into the water. From here, the operator has to control the 6 degrees of freedom of the AUV to find the pipeline. The operator uses two camera views along with other sensor data as listed in the User Interface section, to control the vehicle. The degree of freedom and their controls are as follows: The AUV is is aligned with the pipeline and then follows it to find the leak. When the leak is visible in the downwards facing camera, we consider the leak to be found and proceed towards gates suspended in the water, to better test the controls of the vehicle. Once we have crossed all the four gates, we consider the mission as success.

Table 1. Controls

Degree of Freedom	Positive	Negative
Roll	L	J
Pitch	K	I
Yaw	E	Q
Surge	W	S
Sway	D	A
Heave	Left Shift	Space

The vehicle experiences buoyant force according to its submerged volume. This is achieved by dividing the boat mesh into triangles and then calculating the number of triangles which are submerged, whether fully or partially, underwater. An upward force is applied on individual triangles and the cumulative effect of those forces is seen as the buoyant force on the body. But to use this force, we need a constant downward force from the vehicle's thrusters to keep the AUV stabilized at a depth, which can only be achieved after ROS integration. So to get the desired effect when using Unity3D alone, gravity is switch off.

In the ROS model, we will also have automated sink, automated return to the base station functions and coordinate tagging of the leak along with various other functionalities.

4. UNDERWATER ENVIRONMENT

Along with functionality, aesthetics also play a major role in the simulator. We have created the seabed using simplex noise and custom built shaders are used to simulate the underwater environment. Eventually, we will be able to adjust underwater lightning to simulate various depths and thus use the vehicle's illuminating lights to explore the seabed.

5. BLENDER

All the Prefabs of the simulation were modelled using blender. This served as a great leaning experience in both the blender environment and Unity3D. Objects are exported as Filmbox (.fbx) file type from blender, so they can be imported into Unity3D. The objects behave differently in Unity once they are imported and therefore a little care is needed when using them. The major difference comes in the way the shaders built in blender behave in unity, so sometimes new shaders were needed after the objects were imported into Unity.

6. ASSETS AND ENVIRONMENTS

This section will list the major assets identified at this point of time to be used to build the simulation environment.

1. Boat control Station
2. Unmanned Underwater Vehicle
3. Water Container
4. Water
5. Gates/Checkpoints
6. Terrain
7. Material, Textures and Effects

7. FORCES

The forces to be considered for the simulation are:

- Gravity
- Drag
- Angular Drag
- Thrust
- Buoyancy
- Collisions

Some of these forces will be taken care of by the Rigidbody plugin of Unity3D, and others will be defined by C# scripts.

8. SENSORS

The sensors to be deployed on the UUV are as follows:

- Two camera- Front facing and Downwards facing.
- Inertial Measurement Unit (IMU)
- Pressure Sensor
- Temperature Sensor
- Proximity Sensor
- Compass
- Navigation Lights

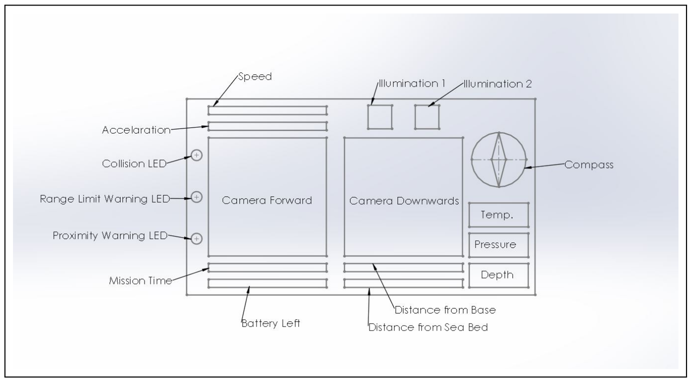

Fig. 1. User Interface for NUSim outlining the elements of the interface.

9. USER INTERFACE

The User Interface (UI) allows the operator to monitor the state of the vehicle and help them to control the UUV. A basic layout of the UI was prepared as shown in *Figure 1*.

10. SCRIPTS

The simulation is made possible by the various scripts that control the whole process. Those scripts are listed below.

1. AUVControl
2. Water and Underwater Effects
3. Triangle mesh data
4. Underwater Mesh Identifier
5. Calculations for Underwater Mesh
6. Buoyant Force
7. Coordinates Calculator
8. Coordinates Marking
9. Surface
10. Collision
11. Buoyant Force
12. Sensor data
13. Relative Distance
14. Speed and Acceleration
15. Range Limit
16. Proximity Warning
17. Compass
18. Illumination
19. Mission Time and Battery left

and various other general and UI scripts which will be identified during the course of development.

11. AIM FOR VERSION 2

Some features that I don't intend to include in the first iteration of the simulation but want to include in the second iteration:

- Camera Movement and Zoom Controls
- Send Sensor data to ROS and save via rosbag
- Simple teleoperation commands using ROS
- Mark the coordinates of the leak and save its picture.

  This list will be updated when I will encounter problems during the development process and when new ideas might cross my mind.