# AVALIAÇÃO DO SOFTWARE UNITY EM SIMULAÇÕES SUBAQUÁTICAS

**Thâmara Raíssa Monteiro Lins**[1]; Felipe Mohr Santos Muniz Barreto[2]; Jean Paulo Silva[3]; Rodrigo Fonseca Carneiro[4]
[1] Graduando em Engenharia Mecânica; Iniciação científica – Robótica; thamara.lins@fbter.org.br
[2] Centro Universitário SENAI CIMATEC; Salvador-BA; felipe.barreto@fieb.org.br
[3] Centro Universitário SENAI CIMATEC; Salvador-BA; jean.silva@fieb.org.br
[4] Centro Universitário SENAI CIMATEC; Salvador-BA; rodrigo.carneiro@fieb.org.br

## RESUMO

A utilização do *software Unity* como simulador subaquático traz algumas adversidades com a estabilidade física, mas com o uso do método de *Update* fixo, a precisão e a estabilidade dos resultados se aproximam do esperado. Isso torna esse método extremamente importante para simulações que exigem alta fidelidade física. O *Unity* é uma escolha confiável para situações em que a simulação física é essencial e destaca sua importância para projetos de simulação subaquática.

**PALAVRAS-CHAVE:** Simulação subaquática; *Unity*; Simulador; Precisão física.

## 1. INTRODUÇÃO

Os simuladores são essenciais para o avanço da robótica, fornecendo um ambiente virtual para testar e aprimorar algoritmos, estratégias de controle e design de hardware.[1] Na robótica, onde as simulações em ambientes reais possuem recursos financeiros elevados e arriscados, os simuladores oferecem uma solução eficaz para aprimorar e validar sistemas robóticos em um ambiente virtual antes da implementação no mundo real.[1]

Dentro do campo de simuladores subaquáticos, existem diversos *softwares* que se destacam por sua robustez e versatilidade. Dentre elas, os simuladores *Gazebo* e *CoppeliaSim*, além das *engines Unity* e *Unreal Engine*, entre outros, oferecendo recursos e aplicações para as simulações. Essas plataformas são utilizadas em uma variedade de contextos, desde analisar o comportamento de um robô em um ambiente dinâmico até a pesquisa de novos métodos de exploração.

Neste trabalho, será utilizado o *Unity* como simulador subaquático. Desenvolvido pela *Unity Technologies*, o *Unity*, é um *software* criado inicialmente para o desenvolvimento de jogos em diversas plataformas, mas também se destaca pela sua finalidade no desenvolvimento de simulações interativas, devido a sua ampla gama de recursos e ferramentas contendo modelagem 3D, animação, física, iluminação e interação do usuário.[2] Ao utilizar sua aplicação para o ambiente subaquático, almeja-se explorar suas capacidades em simular fenômenos e comportamentos específicos dos ambientes aquáticos.

O objetivo deste estudo é empregar o *Unity* para simular ambientes subaquáticos, analisando sua capacidade na representação de fenômenos físicos simples e comportamentos típicos desses contextos subaquáticos para avaliar sua viabilidade e adequação para integração em simulações subaquáticas mais complexas.

## 2. METODOLOGIA

Esta pesquisa tem como finalidade a análise do impacto da utilização do *software Unity* nas simulações de física para robótica subaquática. Para alcançar o objetivo do estudo, foram implementadas pesquisas exploratórias, abrangendo não apenas a revisão de fontes bibliográficas, mas também uma verificação de dados com resultados esperados, utilizando conhecimentos de física básica, e comparação com a outra *engine* comumente utilizada para simulação, *Unreal Engine*, utilizando sua versão 4 (UE4).

O *software Unity* utiliza a *engine* de física *PhysX, da Nvidia,* que simula com precisão a colisão e dinâmica de corpos rígido.[3] Ele suporta dinâmica de corpo rígido, dinâmica de corpo mole, controladores de personagens, dinâmica de veículos, partículas e simulação de fluido volumétrico.[3]

Para avaliar o desempenho da física no *software* em utilização, foram realizados testes aplicando a força de empuxo em um corpo rígido. As métricas utilizadas para analisar esse resultado foram força, posição, velocidade e aceleração. Foram utilizados os conceitos da 2ª Lei de Newton (Equação 01) e da equação do Movimento Retilíneo Uniforme, utilizando a função de posição em relação ao tempo (Equação 02).

Equação 01: $F = m \cdot a$

Equação 02: $S = S_0 + V_0 + \frac{at^2}{2}$

Onde: $F$ representa a Força em N (Newton), $m$ é a massa em $Kg$ do corpo rígido. $S$ e $S_0$ representam sua posição final e inicial, respectivamente, ambas em m (metro). $V_0$ é a velocidade inicial em $m/s$, $t$ representa o tempo em s (segundos) e $a$ representa a aceleração do corpo em $m/s^2$.

A aplicação de uma força constante resulta em uma aceleração constante. Com isso foi possível determinar a aceleração do corpo rígido calculando a força resultante, ou seja, a diferença entre a força de impulso e a força peso do corpo rígido (Equação 03). A partir da aceleração, podemos proceder ao cálculo da posição e velocidade do corpo rígido.

Equação 03: $F_{res} = F_E - F_P$

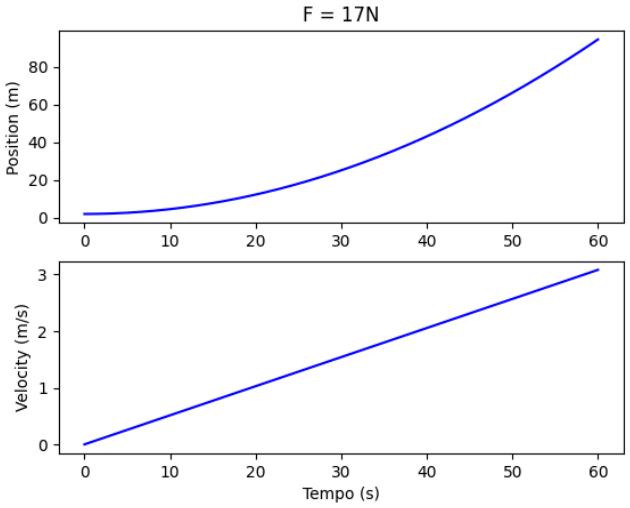
Equação 04: $F_E = Vol \cdot d \cdot g$

Equação 05: $F_p = m \cdot g$

Onde: $F_{res}$, $F_E$, $F_P$representam a força resultante, força de empuxo e força peso, respectivamente, ambas em Newton. $Vol$ é o volume em $m^3$ do corpo rígido, $d$ é a densidade do fluido em $Kg/m^3$ e $g$ é a aceleração da gravidade em $m/s^2$.

O gráfico a seguir representa o comportamento esperado da posição em relação ao tempo e da velocidade em relação ao tempo, os quais serão utilizados como parâmetros nos testes das engines. Além disso, espera-se que a aceleração seja constante no valor de 0.0513 $m/s^2$, para o corpo rígido utilizado de massa de 350 $Kg$ e volume 0.342251812 $m^3$.

Gráfico 01: Posição x tempo e Velocidade x tempo



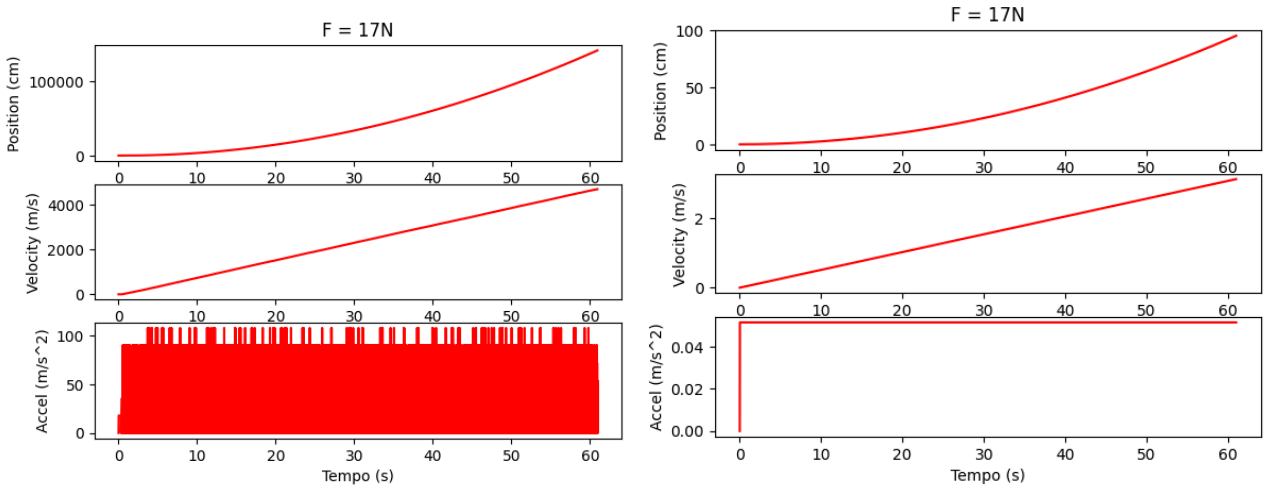

Fonte: Autoria Própria.

## 3. RESULTADOS E DISCUSSÃO

Os resultados do teste utilizando o *software Unity* inicialmente não foram favoráveis, pois a engine não alcançou os resultados esperados. A presença de ruídos e a disparidade entre os resultados que foram obtidos e os que foram previstos são resultados da ausência de um método de *Update* fixo para a simulação, o qual é fundamental para assegurar a estabilidade da física ao longo do intervalo de tempo.[4] Este método é baseado em atualizações com uma taxa de quadros fixa, garantindo que a diferença de tempo entre cada um dos *updates* da simulação seja sempre o mesmo.[4]

Com a utilização do método de *FixedUpdate* do *Unity*, a aplicação de forças no corpo rígido obtém resultados semelhantes ao esperado. Ao realizar o teste utilizando o método de *Update* fixo, os resultados foram favoráveis.[4] A utilização de um método de *Update* fixo para a simulação garantiu estabilidade da física através da aplicação de forças no corpo rígido, obtendo resultados semelhantes ao esperado. Como pode ser observado no gráfico 02 abaixo.

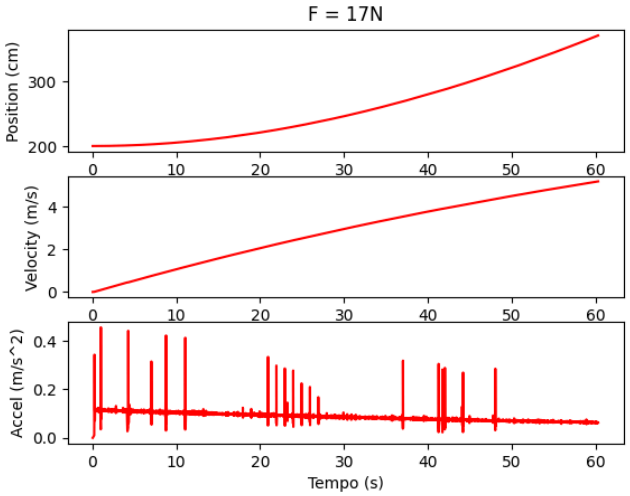
Gráfico 02: Posição x tempo; Velocidade x tempo; Aceleração x tempo *Unit* sem *Update* fixo (esquerda) e com *Update* fixo (direita)




Fonte: Autoria Própria.

Ao realizar a comparação com a engine da *Unreal Engine* 4, devido a sua mesma engine de física,[4] foi notado que os resultados do teste utilizando o UE4 não foram favoráveis, a *engine* não alcançou os resultados esperados perante a física, além de apresentar ruídos na sua aceleração e a mesma encontra-se com a aceleração diminuindo, como pode ser observado no Gráfico 03. Neste *software* não foi possível utilizar o método do Update fixo devido a limitações da *engine*.

Gráfico 03: Posição x tempo; Velocidade x tempo; Aceleração x tempo UE4




Fonte: Autoria Própria.

## 4. CONSIDERAÇÕES FINAIS

O *software Unity* se ressalta como uma escolha para simulações que demandam alta fidelidade física, apresentando notável desempenho nesse aspecto quando utilizado o método de *Update* fixo. Com isso, a utilização de outro *software* de simulação, como o *Unity*, destaca-se como uma opção para cenários onde a simulação física é um requisito essencial devido a sua estabilidade e precisão alcançadas.

A comparação com o *Unreal Engine* 4 demonstrou limitações significativas em relação à estabilidade física. Devido a esse fator, a utilização do *Unity* surge como uma opção sólida para cenários que exigem a simulação física, sendo eficiente para atender às demandas dessas simulações.

## 5. REFERÊNCIAS

[1]COLLINS, Jack et al. **A review of physics simulators for robotic applications**. IEEE Access, v. 9, p. 51416-51431, 2021.

[2] Unity3d. Disponível em: http://unity3d.com.

[3] UNITY DOCUMENTATION. **Unity User Manual 2023.1 (alpha)**. 2023. Disponível em: https://docs.unity3d.com/2023.1/Documentation/Manual/index.html.

[4]**NVIDIA DEVELOPER**. PhysX, s.d. Disponível em: https://developer.nvidia.com/physx-sdk.