

Trabalho de Modelagem e Simulação
Aluno: Pedro Henrique de Brito Canabarro

Para este trabalho, irei abordar a pesquisa em dois softwares: o MATLAB e o Unreal Engine.

1. MATLAB

a) Descrição Geral do MATLAB:

O MATLAB é uma plataforma de programação e computação numérica de alto nível utilizada por milhões de engenheiros e cientistas para analisar os dados, desenvolver algoritmos e criar modelos. Sua linguagem é baseada em matrizes, o que a torna ideal para a resolução de problemas matemáticos e técnicos.

O Simulink é um ambiente de diagrama de blocos integrado ao MATLAB, utilizado para modelagem, simulação e análise de sistemas dinâmicos multidomínio. Ele permite desenhar modelos como diagramas de blocos, o que facilita a visualização e a compreensão de sistemas complexos.

Áreas de aplicação:

- Engenharia de controle e automação.
- Processamento de sinais e comunicações.
- Modelagem de sistemas físicos (mecânicos, elétricos, térmicos).
- Finanças computacionais e econometria.
- Desenvolvimento de sistemas.

b) Classificação

O MATLAB/Simulink pode ser classificado como uma ferramenta multimétodo, pois suporta a modelagem de diversos tipos de sistemas:

- **Sistemas Contínuos:** É seu principal foco, permitindo a solução de equações diferenciais ordinárias para modelar sistemas físicos que mudam continuamente ao longo do tempo (Ex: dinâmica de um veículo, circuitos elétricos).
- **Sistemas de Eventos Discretos:** Por meio de blocos específicos no Simulink, é possível modelar sistemas cujo estado muda em pontos discretos no tempo (Ex: filas de atendimento, logística de produção).
- **Baseado em Agentes:** Embora não seja sua função própria, a modelagem baseada em agentes pode ser implementada com toolboxes específicas (como a Reinforcement Learning Toolbox) ou através de programação direta no MATLAB para simular interações entre agentes autônomos.

c) Licenciamento

O MATLAB é um software comercial e pago. O licenciamento é modular, ou seja, o usuário adquire a licença base do MATLAB e pode adicionar caixas de ferramentas específicas para diferentes áreas. Existem diferentes tipos de licença:

- **Comercial:** Para uso corporativo.
- **Acadêmica:** Licenças com desconto para universidades e instituições de ensino.

- **Estudante:** Versões de baixo custo para estudantes.
- **Home:** Para uso pessoal e não comercial.

Não é um software open-source.

d) Recursos de IA

O MATLAB possui uma integração muito forte e robusta com Inteligência Artificial e Machine Learning através de toolboxes dedicadas:

- **Deep Learning Toolbox:** Permite projetar, treinar e analisar redes neurais profundas.
- **Machine Learning Toolbox:** Oferece algoritmos e aplicativos para treinar, visualizar e avaliar uma ampla gama de modelos de aprendizado de máquina.
- **Reinforcement Learning Toolbox:** Fornece funções e blocos para treinar políticas de decisão (agentes) usando aprendizado por reforço, muito utilizado em robótica e sistemas de controle autônomo. Esses recursos podem ser diretamente integrados aos modelos do Simulink, permitindo por exemplo, que um agente de IA controle um sistema dinâmico simulado.

e) Aplicação Prática

Exemplo Real de Uso:

A indústria automotiva utiliza extensivamente o MATLAB/Simulink para o desenvolvimento de sistemas de controle veicular, como freios ABS, controle

de estabilidade (ESC) e, mais recentemente, para projetar e simular os algoritmos de percepção e decisão de veículos autônomos. Os modelos simulam o comportamento do veículo em diferentes condições de pista e trânsito antes mesmo de um protótipo físico ser construído.

Exemplo de Simulação Simples (Código):

Abaixo está um exemplo de código em MATLAB que simula e plota o movimento de um pêndulo simples, um problema clássico de sistema contínuo resolvido com uma equação diferencial ordinária (EDO).

```
clear all;
clc;

g = 9.81;
L = 1.0;
t_span = [0 10];
theta0 = [pi/4, 0];

[t, theta] = ode45(@(t,theta) pendulo_ode(t, theta, g, L), t_span, theta0);

figure;
plot(t, theta(:,1), '-o', t, theta(:,2), '-r');
title('Simulação de um Pêndulo Simples');
xlabel('Tempo (s)');
ylabel('Ângulo (rad) e Velocidade Angular (rad/s)');
legend('Ângulo (\theta)', 'Velocidade Angular (\omega)');
grid on;
```

Explicação do Código:

1. `pendulo_ode` é uma função que define a EDO do pêndulo.
2. O script principal define os parâmetros (gravidade, comprimento, etc.) e as condições iniciais (posição e velocidade).
3. `ode45` é o solver do MATLAB que resolve a EDO numericamente ao longo de um intervalo de tempo.
4. Ao final, o código plota a posição angular do pêndulo ao longo do tempo.

2. Unreal Engine

a) Descrição Geral do Unreal Engine

A Unreal Engine (UE) é um motor de jogo e uma ferramenta de criação 3D em tempo real desenvolvida pela Epic Games. Originalmente criada para o desenvolvimento de jogos, sua capacidade de renderização fotorrealista, simulação de física precisa e escalabilidade a tornaram uma ferramenta poderosa para simulações interativas em diversas áreas fora do entretenimento.

Principais áreas de aplicação (para simulação):

- **Simulação de Veículos Autônomos:** Criação de ambientes virtuais (cidades, estradas) para treinar e validar algoritmos de IA de direção.
- **Digital Twins (Gêmeos Digitais):** Criação de réplicas virtuais de sistemas físicos, como fábricas ou cidades, para monitoramento, análise de cenários e otimização.
- **Simulação para Robótica:** Teste de robôs em ambientes virtuais antes de sua implementação física, especialmente para tarefas de manipulação e navegação.
- **Visualização Arquitetônica e Treinamento:** Criação de experiências imersivas para visualizar projetos de engenharia ou treinar operadores de máquinas pesadas em um ambiente seguro.

b) Classificação

A Unreal Engine é uma plataforma multimétodo, com forte ênfase em:

- **Baseado em Agentes:** É sua principal característica para simulação. Cada personagem, veículo ou robô pode ser um “agente” com seu próprio comportamento, percepção e lógica de decisão (definidos via código C++ ou pelo sistema de scripting visual Blueprints).
- **Sistemas Contínuos:** A UE possui um motor de física robusto (Chaos Physics) que simula continuamente fenômenos como gravidade, colisões, dinâmica de fluidos e materiais deformáveis.
- **Eventos Discretos:** Embora a simulação principal seja em tempo real, a lógica de controle é frequentemente baseada em eventos (Ex: “ao colidir com o objeto X, faça Y”, “ao entrar no volume Z, ative a luz”).

c) Licenciamento

A Unreal Engine possui um modelo de licenciamento bastante acessível

- **Gratuita para uso:** O download e o uso do motor são gratuitos.
- **Modelo de Royalties (para jogos):** Para jogos comercializados, a Epic Games cobra 5% de royalties sobre a receita bruta após o primeiro milhão de dólares.
- **Uso gratuito para outras áreas:** Para projetos que não são jogos, como simulações, filmes e projetos de arquitetura, geralmente não há cobrança de royalties, tornando-a uma opção muito atraente para fins acadêmicos e de pesquisa.
- **Código-fonte aberto (*"source-available"*):** O código-fonte completo em C++ está disponível publicamente (via GitHub), permitindo modificações

e compilações personalizadas. Tecnicamente, não é *open-source* por ter restrições de licença, mas é aberto para consulta e modificação.

d) Recursos de Inteligência Artificial

A UE vem com ferramentas de IA prontas para uso, focadas em comportamento de agentes em jogos e simulações:

- **Behavior Trees (Árvores de Comportamento):** Ferramenta visual para modelar a lógica de decisão de um agente de forma hierárquica e complexa (ex: patrulhar, perseguir um alvo, fugir).
- **AI Perception:** Componente que permite aos agentes perceberem o mundo ao seu redor através de "sentidos" virtuais como visão, audição e tato.
- **Environment Query System (EQS):** Permite que um agente faça "perguntas" sobre o ambiente para tomar decisões inteligentes (ex: "Qual é o melhor local para se esconder?").

Além disso, a UE pode ser **integrada com frameworks externos de Machine Learning** como TensorFlow e PyTorch através de plugins, permitindo que modelos de ML treinados em Python controlem agentes dentro da simulação.

e) Aplicação Prática

Exemplo Real de Uso:

O projeto **AirSim**, desenvolvido pela Microsoft, é um plugin de código aberto para a Unreal Engine. Ele é usado para criar simulações de alta fidelidade de drones e carros. Pesquisadores o utilizam para gerar dados sintéticos (imagens

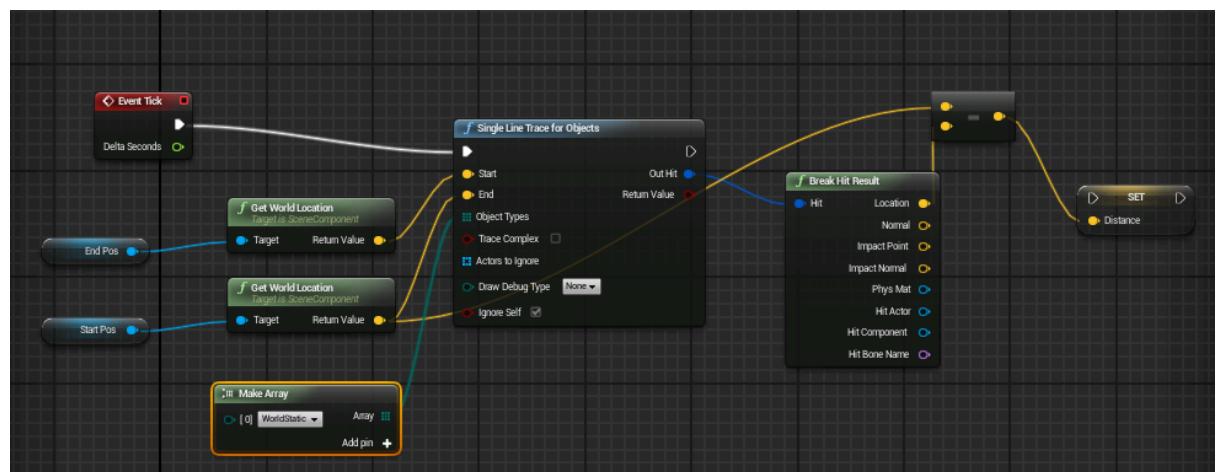
de câmera, leituras de sensores) para treinar modelos de deep learning para condução autônoma, sem o risco e o custo de testes no mundo real.

Exemplo de Simulação Simples (Blueprints):

Uma simulação simples na UE seria criar um semáforo que controla o fluxo de veículos. Isso pode ser feito inteiramente com **Blueprints**, o sistema de scripting visual da UE.

Explicação do Conceito:

- **Autor Semáforo (Blueprint):** Cria-se um "Aotor" para o semáforo. Dentro dele, uma lógica de temporizador (Timer) alterna o estado das luzes (vermelha, amarela, verde) a cada X segundos. Ele também contém um volume de colisão (trigger volume) na frente da linha de parada.
- **Autor Veículo (Blueprint):** Cria-se um "Aotor" para o carro. Ele terá uma lógica para se mover para frente constantemente.
- **Lógica de Interação:** O carro usa um "raycast" (um raio invisível) para detectar o que está à sua frente. Se o raio detectar o semáforo e a luz estiver vermelha, a velocidade do carro é definida como zero. Se a luz estiver verde, sua velocidade retorna ao normal.



```
// Wrapper around WeaponTrace to handle multiple hits
FHitResult DoseSingleBulletTrace(const FTransform& Transform, const FVector2D ImpactPoint, const FVector ImpactNormal, const FCollisionChannel Channel, const FCollisionResponseParams ResponseParams, const FCollisionObjectQueryParams QueryParams, const FCollisionResponseParams& OutResponseParams) const
{
    // Traces all of the bullets in a single cartridge
    void TraceBulletsInCartridge(const FTransform& Transform, const FVector2D ImpactPoint, const FVector ImpactNormal, const FCollisionChannel Channel, const FCollisionResponseParams ResponseParams, const FCollisionObjectQueryParams QueryParams, const FCollisionResponseParams& OutResponseParams) const
    {
        virtual void AddAdditionalTraceIgnore(FCollisionResponseParams& ResponseParams, const FCollisionObjectHandle CollisionObjectHandle, const FCollisionObjectHandle TraceObjectHandle) const
        {
            // Determine the trace channel to use
            virtual ECollisionChannel DetermineTraceChannel() const
            {
                void PerformLocalTargeting(OUT TArrray< FHitResult> FoundHits)
                {
                    FVector GetWeaponTargetingSourceLocation();
                    FTransform GetTargetingTransform(APawn* Target);
                    void OnTargetDataReadyCallback(const FGameplayAbilityTargetDataHandle& TargetDataHandle);
                    void StartRangedWeaponTargeting();
                    // Called when target data is ready
                    UFUNCTION(BlueprintImplementableEvent)
                    void OnRangedWeaponTargetDataReady(const FGameplayAbilityTargetDataHandle& TargetDataHandle);
                    private:
                    FDelegateHandle OnTargetDataReadyCallback;
                }
                if (FoundHits.Num() > 0)
                {
                    const int32 CartridgeID = FMath::Rand();
                    for (const FHitResult& FoundHit : FoundHits)
                    {
                        FLYRAGameplayAbilityTargetData_SingleTargetHit* NewTargetData = new FLYRAGameplayAbilityTargetData_SingleTargetHit();
                        NewTargetData->HitResult = FoundHit;
                    }
                }
            }
        }
    }
}
```

Fonte: fotos retiradas da internet.