UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

Lucas Porto de Almeida

Trabalho de Conclusão de Curso

Desenvolvimento de um gêmeo digital de um sistema didático modular de manufatura

Campina Grande - PB

Lucas Porto de Almeida

Desenvolvimento de um gêmeo digital de um sistema didático modular de manufatura

Trabalho de Conclusão de Curso submetido à Coordenação de Graduação em Engenharia Elétrica da Universidade Federal de Campina Grande, Campus Campina Grande, como parte dos requisitos necessários para obtenção do título de Graduado em Engenharia Elétrica.

Universidade Federal de Campina Grande - UFCG

Centro de Engenharia Elétrica e Informática - CEEI

Departamento de Engenharia Elétrica - DEE

Orientador: Rafael Bezerra Correia Lima, D.Sc.

Campina Grande, Brasil

15 de dezembro de 2020

Lucas Porto de Almeida

Desenvolvimento de um gêmeo digital de um sistema didático modular de manufatura

Trabalho de Conclusão de Curso submetido à Coordenação de Graduação em Engenharia Elétrica da Universidade Federal de Campina Grande, Campus Campina Grande, como parte dos requisitos necessários para obtenção do título de Graduado em Engenharia Elétrica.

Trabalho aprovado em: / /2020

Rafael Bezerra Correia Lima, D.Sc.
Orientador

George Acioli Júnior, D.Sc.
Convidado

Campina Grande, Brasil
15 de dezembro de 2020

Dedico este trabalho aos meus amados pais, Alexandre Santana de Almeida e Leni Porto de Almeida.

Agradecimentos

Em primeiro lugar, agradeço aos meus pais, Alexandre e Leni, por todo o apoio, incentivo e encorajamento, assim como por todo o tempo e esforços investidos em mim durante todos esses anos. Esse suporte foi o que me permitiu chegar até aqui.

Agradeço a toda a minha família e, em especial, aos meus tios Alberto, Augusto, Denise e Wagner, por investirem na minha educação e por sempre me incentivarem a ir mais longe.

Agradeço ao professor Rafael Bezerra por ter me recebido no Laboratório de Instrumentação Eletrônica e Controle (LIEC), por ter concordado com a proposta deste trabalho e em ser o meu orientador.

Agradeço ao professor George Acioli, por todas as sugestões, dicas e conselhos com relação a este trabalho, assim como por possibilitar a sua utilização na disciplina de Sistemas de Automação Industrial.

Agradeço ao professor Péricles Barros, por me aceitar no LIEC, por acreditar na relevância do tema, por todo o incentivo e por toda a orientação no laboratório.

Agradeço aos meus companheiros de laboratório, Breno, Egydio, Matheus e Ravi por todos os momentos de diversão e companheirismo. Agradeço de forma especial a Victor e Vinícius, por terem se disposto a fazer os primeiros testes com a aplicação e pelo incentivo.

Agradeço ao meu amigo Luquinhas, pela companhia nas aulas de Alemão, por toda a disponibilidade, suporte e incentivo, assim como por todos os momentos memoráveis durante todos esses meses.

Por fim, agradeço ao meu amigo Júnior, por toda a orientação desde o primeiro dia no laboratório, por todas as conversas, por todo o apoio, por toda a disponibilidade, por todas as caronas, por toda a tranquilidade transmitida e por ter me recomendado ao professor Rafael, para que fizesse parte da equipe do LIEC. Serei sempre grato por esta incrível oportunidade.

"Deus é o nosso refúgio e a nossa fortaleza,
auxílio sempre presente na adversidade."
Salmos 46:1

Resumo

O gêmeo digital (GD) é uma réplica virtual de um ativo físico, o qual é capaz de oferecer uma representação gráfica tridimensional (3D) em tempo real dos componentes, dinâmica e dados associados a este ativo, constituindo-se, dessa forma, como uma ferramenta com grande potencial de aplicação em diversas áreas, especialmente no contexto educacional. No presente trabalho são apresentados os passos relativos ao desenvolvimento de um GD, inserido em uma aplicação, de um sistema didático modular de manufatura presente no Laboratório de Instrumentação Eletrônica e Controle (LIEC) da Universidade Federal de Campina Grande (UFCG), criado com o objetivo de auxiliar no desenvolvimento de projetos de automação industrial. A aplicação foi desenvolvida no motor de jogos Unity, utilizando, de maneira integrada, recursos de realidade aumentada e um cliente OPC UA para exibir os componentes 3D do sistema virtual, modelados no *software* AutoCAD, e permitir a comunicação em tempo real com o sistema físico.

Palavras-chaves: Gêmeo Digital, Realidade Aumentada, Indústria 4.0, OPC UA.

Abstract

The digital twin (DT) is a virtual replica of a physical asset, which is capable of providing a real-time three-dimensional (3D) graphical representation of the components, dynamics and data associated with this asset, thus constituting itself as a tool with great potential for application in many areas, especially in the educational context. The present work presents the steps related to the development of a DT, inserted in an application, of a modular didactic manufacturing system present in the Laboratory of Electronic Instrumentation and Control (LIEC) of the Federal University of Campina Grande (UFCG), created with the objective of assisting in the development of industrial automation projects. The application was developed on the Unity game engine, using, in an integrated way, augmented reality resources and an OPC UA client to display the 3D components of the virtual system, modeled in AutoCAD software, and allow real-time communication with the physical system.

Key-words: Digital twin, Augmented Reality, Industry 4.0, OPC UA.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

3D	Tridimensional
DLL	*Dynamic-Link Library*
GD	Gêmeo Digital
LIEC	Laboratório de Instrumentação Eletrônica e Controle
OLE	*Object Linking and Embedding*
OPC	*OLE for Process Control*
QR	*Quick Response*
RA	Realidade Aumentada
RV	Realidade Virtual
SDK	*Software Development Kit*
UA	*Unified Architecture*
UFCG	Universidade Federal de Campina Grande

Sumário

1 Introdução

A indústria de manufatura moderna lida com ativos de diferentes graus de complexidade, os quais requerem funcionários responsáveis por seu monitoramento, manutenção e operação, distribuídos em diversos estágios do ciclo de produção, dos quais é esperado conhecimento técnico, precisão e eficiência.

O treinamento destes envolve custos relativos ao manejamento de treinadores e ao tempo despendido, o que atrai a busca por ferramentas capazes de oferecer, aos funcionários, auxílio na compreensão mais rápida e independente dos ativos a fim de poupar custos e ocupação de trabalho especializado associados a processos de treinamento.

O *software* de desenho assistido por computador, ou CAD (do inglês *Computer Aided Design*), desenvolvido a partir de meados da década de 60, a partir do qual era possível criar desenhos técnicos simples em 2D, permite atualmente desenvolver modelos tridimensionais (3D) de alta fidelidade, de diversos tipos de estruturas, ambientes e sistemas. A estes podem ser integradas um conjunto de informações relativas às suas, restrições físicas, materiais e dados proprietários.

À medida que a qualidade da modelagem gráfica dos componentes 3D evoluía, passou-se também a haver a busca por modelar a dinâmica e o comportamento do sistema, obedecendo as restrições físicas e os materiais utilizados, o que permitiu o começo da integração de simulações e testes aos programas com *software* CAD.

Tal integração, nestes programas, se tornou atrativa ao passo que possibilitava visualizar os componentes 3D, de alta resolução, com animações físicas realistas das dinâmicas do sistema, em execução nas simulações cujos resultados aproximavam-se dos reais. Permitindo, assim, visualizar, simular e controlar um sistema virtual que cada vez mais se configurava como reflexo do sistema real.

Com o intuito de formalizar esta abordagem, em 2002 o conceito e modelo do Gêmeo Digital (GD) foram publicamente introduzidos, mas assumindo este termo apenas em 2010. Essencialmente, o conceito refere-se à ideia de que um construto digital de informações sobre um sistema físico poderia ser criado como uma entidade por si só. Este construto digital seria um "gêmeo" (em inglês, *twin*) do sistema físico, na medida que permitisse obter quaisquer informações que se obteriam inspecionando o gêmeo físico. (GRIEVES, 2016).

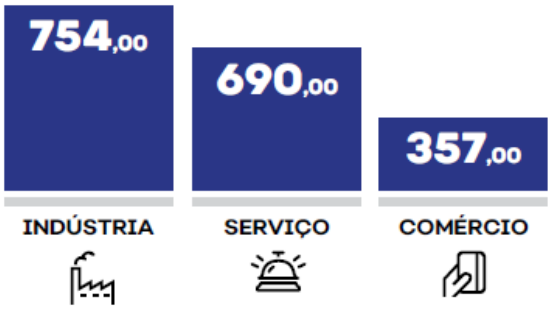
1.1 Motivação

Atualmente, considera-se que uma das principais necessidades do setor industrial de manufatura é poupar trabalho especializado, o qual está associado a um custo considerável e possui baixa disponibilidade (BUCSAI et al., 2020).

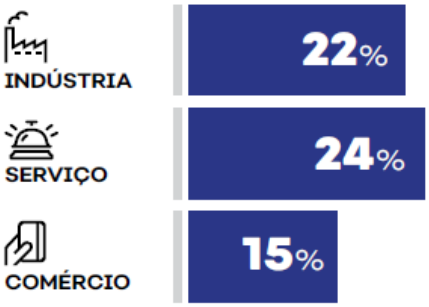
Um conjunto desses trabalhadores especializados e experientes são designados para a realização, gerenciamento e supervisão de treinamentos, o que se reflete no fato de que o setor industrial, no cenário brasileiro, é o setor que mais possui gastos associados a T&D (Treinamento e Desenvolvimento), seja por colaborador, seja em números absolutos, como pode se observar na Figura 1a, além de não ser o setor que mais utiliza tecnologias para a realização de treinamentos, como pode se constatar a partir dos dados presentes na Figura 1b.

Figura 1 – Investimentos em T&D das empresas brasileiras.

(a) Investimento anual em T&D das empresas brasileiras por colaborador.



(b) Percentagem dos treinamentos que são realizados por meio de tecnologia.



Fonte: Adaptado de Panorama do Treinamento no Brasil 2019/2020.

Os recursos de Realidade Aumentada (RA) atualmente oferecem a possibilidade de serem implementados e integrados em aplicações de relativo baixo custo computacional e alta taxa de quadros por segundo, os quais permitem a imersão do usuário em um cenário que possui informações agrupadas do espaço real e do espaço virtual.

Especificamente no contexto de treinamento na indústria, os recursos de RA oferecem a oportunidade de criação de treinamentos mais eficazes, reduzindo significativamente os custos associados e riscos de segurança possivelmente presentes na medida em que permitem treinamentos em cenários virtuais que replicam os reais de tal forma que os trabalhadores podem ser capacitados e adquirir confiança como se estivessem lidando com os sistemas reais de fato.

A Festo é uma empresa alemã de destaque no setor de automação industrial. Sua linha de produtos didáticos, é amplamente utilizada em treinamento vocacional industrial, pesquisa e ensino em vários países. O MPS, (do inglês, *Modular Production System*) é uma linha de produção em escala de laboratório que simula funções que podem ser identificadas

em uma fábrica, como distribuição, teste, processamento, entre outras, sendo cada uma destas presentes, de forma individual, em um módulo da linha (EBEL; PANY, 2006a).

O MPS é utilizado para o treinamento de pessoal em diferentes áreas relacionadas à produção, tais como: planejamento, montagem, programação, comissionamento, operação e manutenção. Os módulos deste sistema podem ser combinados de forma a executar tarefas mais complexas.

Presente no Laboratório de Instrumentação Eletrônica e Controle (LIEC), o sistema modular de manufatura didático, composto pelos módulos de distribuição, separação e classificação do MPS da Festo, por permitir uma montagem, aprendizado, e treinamento independentes, a partir de seus módulos, configura-se como uma planta ideal para o desenvolvimento e validação do trabalho proposto.

1.2 Objetivos

1.2.1 Objetivo geral

Desenvolver um gêmeo digital de um sistema didático modular de manufatura demonstrando as potencialidades relacionadas à sua utilização como ferramenta de ensino.

1.2.2 Objetivos específicos

- Desenvolver modelos em 3D que representem o sistema modular de manufatura didático, presente no laboratório LIEC da UFCG, utilizando o software AutoCAD;
- Implementar a dinâmica dos componentes do sistema em 3D e a interface com o usuário da aplicação no motor de jogos Unity;
- Integrar bibliotecas à solução que permitam a comunicação utilizando o padrão OPC UA entre a aplicação e um servidor OPC UA;
- Integrar recursos de realidade aumentada à aplicação, de forma a apresentar os componentes em 3D a partir de códigos QR;

1.3 Organização do trabalho

O trabalho está organizado em 5 capítulos, incluindo este introdutório, cujo conteúdo é descrito a seguir.

- Capítulo 2: aborda os conceitos teóricos associados aos termos gêmeos digitais e realidade aumentada, os quais embasaram este trabalho, assim como a relevância

do padrão OPC UA e as ferramentas utilizadas para o desenvolvimento do gêmeo digital e da aplicação na qual foi inserido;

- Capítulo 3: descreve os passos relativos ao desenvolvimento do gêmeo digital, desde a modelagem 3D dos componentes no *software* AutoCAD até a integração do modelo físico, dinâmica e comunicação bidirecional por meio de um cliente OPC UA a este, utilizando o *software* Unity;
- Capítulo 4: apresenta os resultados do ponto de vista funcional e de interface com o usuário, relativos ao desenvolvimento realizado no capítulo anterior;
- Capítulo 5: apresenta as conclusões e sugestões de trabalhos futuros.

2 Fundamentacão Teórica

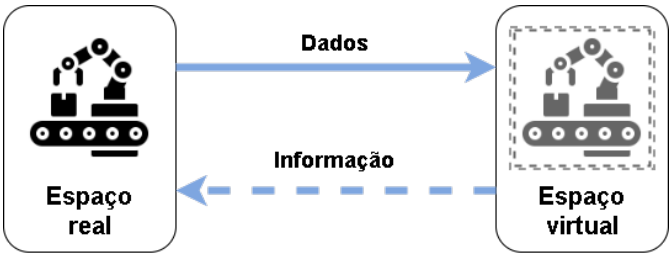
No presente capítulo são apresentados os conceitos e tópicos utilizados como base para este trabalho. Neste sentido, serão abordados o contexto de surgimento e as aplicações dos gêmeos digitais, a evolução dos recursos de realidade aumentada e a proposta do padrão de comunicação OPC UA. Adicionalmente, serão abordadas, de maneira breve, as ferramentas utilizadas para o desenvolvimento da aplicação.

2.1 Gêmeos Digitais

O conceito de um equivalente digital de um produto físico, ou gêmeo digital (GD), foi proposto inicialmente por Grieves em uma de suas aulas do seu curso de Gerenciamento do Ciclo de Vida do Produto, em 2003. Embora o termo "Gêmeo Digital" ainda não houvesse sido associado a este conceito, Grieves o definiu como uma representação virtual do que foi produzido, o qual permite um melhor entendimento das diferenças entre o que foi planejado e o que de fato foi produzido (GRIEVES, 2015).

O modelo proposto para o conceito era constituído essencialmente de três partes: a) produto físico no espaço real, b) produto virtual no espaço virtual e c) os fluxos de dados e informações que conectam ambos os sistemas, como ilustrado no esquemático da Figura 4.

Figura 2 – Esquemático conceitual do GD.



Fonte: Adaptado de (GRIEVES, 2015).

O conceito apresentado por Grieves, o qual tinha como foco o contexto do gerenciamento de produtos de manufatura, foi expandido para outros contextos assim como foram propostas outras definições. Estas definições, propostas ao longo dos anos seguintes, buscaram, por uma parte, tornar o significado do conceito mais abrangente, e por outra, aplicá-lo de forma ainda mais especifica para os demais contextos.

Algumas das definições propostas para o conceito foram as seguintes:

- "O Gêmeo Digital é um conjunto de construções de informações virtuais que descrevem totalmente um produto físico fabricado potencial ou real do nível micro atômico ao nível macro geométrico. Em seu nível ideal, qualquer informação que possa ser obtida na inspeção de um produto fabricado físico pode ser obtido de seu gêmeo digital (GRIEVES; VICKERS, 2017)."
- "Um gêmeo digital é uma simulação multifísica, multiescalar e probabilística integrada de um veículo ou sistema que usa os melhores modelos físicos disponíveis, atualizações de sensores, histórico de frota, etc., para espelhar a vida de seu gêmeo correspondente (GLAESSGEN; STARGEL, 2012)."
- "O Gêmeo Digital é um mapeamento real de todos os componentes do ciclo de vida do produto usando dados físicos, virtuais e dados de interação entre eles (TAO et al., 2018)."
- "uma representação virtual dinâmica de um objeto físico ou sistema em todo o seu ciclo de vida, usando dados em tempo real para permitir a compreensão, aprendizagem e raciocínio (BOLTON et al., 2018)."
- "uma cópia digital do sistema físico para realizar a otimização em tempo real (SÖDERBERG et al., 2017)."
- "Um gêmeo digital é um modelo de computador que espelha e simula um ativo ou sistema de ativos e seu ambiente circundante (COMMISSION et al., 2017)."

É possível, com base nas definições apresentadas, identificar algumas características em comum, tanto com relação ao seu significado do ponto de vista funcional, quanto com relação aos objetivos que se deseja alcançar. Dessa forma, pode-se organizar tais características, de forma estruturada, como apresentado na Tabela 1.

Tabela 1 – Funcionalidades e objetivos em comum dos GDs.

Funcionalidades	**Objetivos**
Modelo físico do ativo	Análise do comportamento físico do ativo sob diferentes condições
Representação virtual dinâmica em 3D	Melhor compreensão dos componentes do ativo e de suas funcionalidades
Comunicação bidirecional em tempo real com o ativo físico	Espelhamento em tempo real dos estados do ativo físico
Simulações de testes e operações	Visualização da dinâmica de operação do ativo em cenários distintos

Fonte: Autoria própria.

Após a compreensão básica das características comuns dos GDs, tomando como referência a tabela apresentada anteriormente, faz-se necessário ainda, do ponto de vista

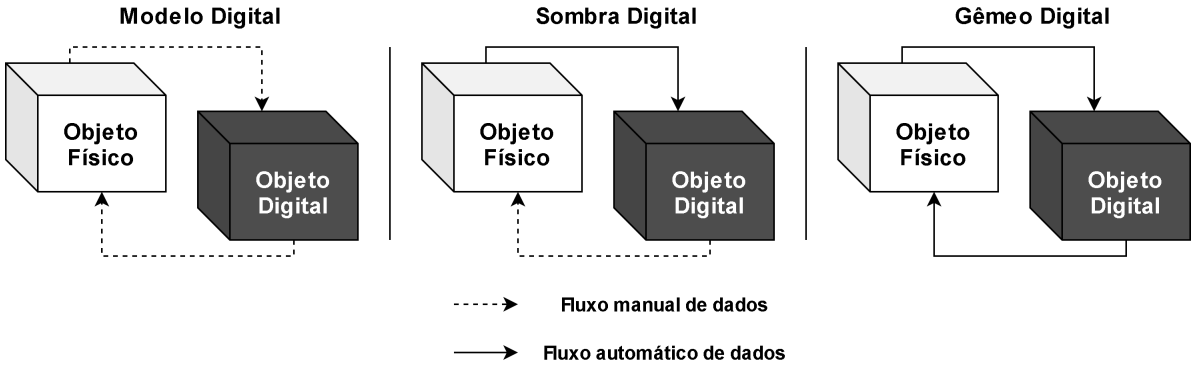
teórico, determinar qual característica distingue os GDs de ferramentas de simulação, dado que ambos possuem elevada similaridade.

Desde a introdução do conceito de GDs, várias implementações deste foram desenvolvidas ao longo dos anos, tomando como base as diferentes definições propostas. Diante disso, alguns autores consideram que a existência de múltiplas soluções e conceitos de GDs, em todos os setores, reflete uma compreensão diversa e incompleta deste conceito (KRITZINGER et al., 2018).

A partir da percepção de que muitas destas implementações não se diferenciavam o suficiente de simuladores já disponíveis, ao ponto de ser possível lhes atribuir o termo de GDs, foi proposta a separação destas em três categorias distintas, com base no aspecto do seu nível de integração (KRITZINGER et al., 2018).

A primeira categoria, Modelo Digital (MD), refere-se a uma representação digital que não possui a capacidade de trocar dados de forma automática com o ativo físico, de tal forma que uma mudança de estado do objeto físico não é capaz de alterar automaticamente o objeto digital, e vice-versa. A segunda categoria, Sombra Digital (SD), por sua vez, refere a uma representação digital que permite a alteração automática do objeto digital a partir da mudança de estado do objeto físico, mas não o inverso. Finalmente, a terceira categoria, Gêmeo Digital (GD), refere-se a uma representação digital que permite a alteração automática do objeto digital a partir da mudança de estado do objeto físico, assim como o inverso. Esse fluxo bidirecional de informações permite, inclusive, que o objeto digital atue como controlador do objeto físico (KRITZINGER et al., 2018).

Figura 3 – Fluxo de dados em um MD, SD e GD.



Fonte: Adaptado de (KRITZINGER et al., 2018).

Neste sentido, identifica-se que é possível distinguir, portanto, os GDs das demais implementações, as quais se encaixam nas duas primeiras categorias propostas, com base na presença da funcionalidade de comunicação bidirecional em tempo real com o ativo físico.

2.2 Realidade Aumentada

A Realidade Aumentada (RA) é uma variação da Realidade Virtual (RV), onde objetos virtuais são sobrepostos ao ambiente real (AZUMA, 1997) e que deve atender a três requisitos essenciais: combinar objetos virtuais e reais em um ambiente real, executar de forma interativa e alinhar, em tempo real, os objetos reais e virtuais (CARDOSO; MARIANO; ZORZAL, 2020).

Por um lado, a RA é capaz de elevar a percepção da realidade, por parte do usuário, a partir da incorporação dos objetos virtuais no ambiente real, enquanto que, por outro lado, a RV substitui o ambiente real por um ambiente virtual, o qual permite uma experiência de imersão ao usuário, fazendo com que este se sinta realmente presente no ambiente simulado (MARTIN; BOHUSLAVA, 2018) (CARDOSO; MARIANO; ZORZAL, 2020).

A partir da identificação do seu potencial como ferramenta tecnológica, esta vem se desenvolvendo de forma rápida nos últimos anos e tem sido aplicada em diversas áreas, tais como jogos, medicina, militar e também na área de manufatura industrial (NEE; ONG, 2013).

O seu uso nesta área se justifica, dentre outros aspectos, pelo fato de que alguns dos problemas críticos no contexto da indústria de manufatura referem-se à simulação, orientação e aprimoramento dos processos de fabricação a serem futuramente lançados, para os quais a tecnologia de RA se mostrou eficaz e inovadora, após o seu amadurecimento (NEE; ONG, 2013).

Figura 4 – Ilustração de uso do óculos de RA HoloLens 2 na indústria.

Fonte: Microsoft.

De forma específica, portanto, a RA é capaz de exibir ao usuário uma quantidade adicional de informações em seu campo de visão, por meio de um dispositivo auxiliar, tal

como um óculos, como mostrado anteriormente, ou como um *smartphone*. Essas informações, por sua vez, referem-se aos componentes, estados ou configurações do objeto de interesse.

Tal visualização de dados, a qual era realizada previamente por meio de relatórios em papel ou gráficos na tela de um computador, pode ser realizada de forma portátil e dinâmica.

2.3 OPC UA

A utilização de sistemas de automação baseados em PC (do inglês, *Personal Computer*), de forma especial na plataforma Windows, e *software* em automação industrial cresceu de forma rápida desde o início da década de 1990. Um dos maiores desafios até então para o desenvolvimento de um *software* de automação padronizado era o acesso a dados de automação em dispositivos que possuíam diferentes protocolos e interfaces (MAHNKE; LEITNER; DAMM, 2009).

Nesse contexto, um grupo de empresas decidiu unir seus esforços no ano de 1995 no sentido do desenvolvimento de um padrão para *drivers* de dispositivos que permitisse um acesso padronizado a dados de automação em sistemas baseados em Windows. No ano de 1996, como resultado desse esforço, surgia a especificação OPC *Data Access*, ou OPC DA, criada e mantida pelo consórcio OPC *Foundation*. Esta especificação foi a primeira do padrão OPC (*OLE for Process Control*), construído com base na tecnologia da Microsoft denominada de OLE (do inglês, Object Linking and Embedding).

A interface OPC DA, portanto, permite a leitura, escrita e monitoramento de variáveis que contém dados atuais dos processos. Grande parte de sua utilização se dá na movimentação de dados, em tempo real, de controladores lógicos programáveis (CLPs) e outros dispositivos de controle para interfaces homem-máquina (IHMs) e outros clientes de exibição destes dados (MAHNKE; LEITNER; DAMM, 2009).

O padrão OPC se tornou, após certo tempo, o padrão universalmente aceito, o qual fornece a capacidade de trocar dados entre diferentes sistemas de automação industrial nas indústrias de manufatura e processo (MAHNKE; LEITNER; DAMM, 2009).

Entretanto, ao longo dos anos, diante dos problemas relativos à utilização mais abrangente do padrão, diversos novos requerimentos por parte das empresas passaram a surgir. Tais requerimentos se referem tanto a aspectos de comunicação entre sistemas distribuídos, quanto a modelagem de dados.

Com o objetivo de atender os requisitos estabelecidos, portanto, definiu-se a necessidade de garantir um grupo de novos parâmetros e funcionalidades. Dentre esses estão a robustez, tolerância a erros e redundância para alta disponibilidade, além da independên-

cia de plataforma e escalabilidade, necessárias para integrar interfaces OPC diretamente a sistemas executando em diferentes plataformas. Ademais, a possibilidade de comunicação via internet através de *firewalls*, segurança e controle de acesso, assim como a interoperabilidade entre sistemas de diferentes fabricantes também fazem parte deste grupo.

A partir do desenvolvimento de tais funcionalidades, foram lançadas em 2008 as especificações do OPC *Unified Architecture*, ou OPC UA. Estas constituem uma arquitetura orientada a serviços e independente de plataforma que integra, em uma estrutura extensível, todas as especificações individuais do OPC Clássico, composto pelas especificações OPC DA, A & E (*Alarm & Events*) e HDA (*Historical Data Access*) (FOUNDATION, 2020).

2.4 Ferramentas utilizadas

Para o desenvolvimento deste trabalho, utilizou-se o *software* AutoCAD em sua versão gratuita para estudantes, do ano de 2018, assim como o motor de jogos multiplataforma Unity, também em sua versão gratuita, do ano de 2018.

2.4.1 AutoCAD

O AutoCAD é um *software* CAD (do inglês, *Computer-Aided Design*), criado pela empresa Autodesk, o qual é utilizado principalmente para a criação de desenhos técnicos em duas dimensões (2D) e modelos tridimensionais (3D) de forma precisa.

Embora a ferramenta seja utilizada, em grande parte, para a criação de esboços em 2D, é possível criar uma diversidade de objetos 3D predefinidos, tais como cilindros, esferas, cunhas, cubos e outros. Por meio de sua interface intuitiva é possível executar comandos, tais como os de extrusão e revolução, para transformar rascunhos 2D em sólidos 3D.

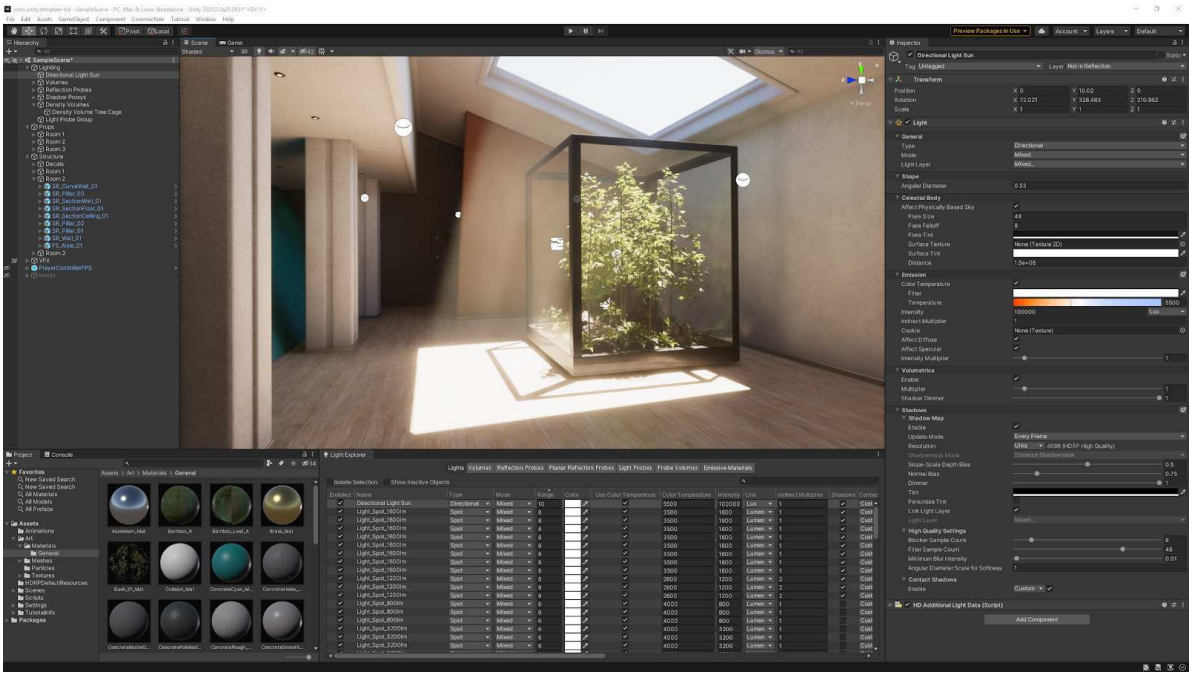
Além da disponibilidade do *software* em sua versão gratuita, a grande variedade de funcionalidades oferecidas, assim como a familiaridade com a ferramenta, outro fator de escolha da versão mencionada foi a capacidade que esta oferece de exportar os objetos 3D desenvolvidos em seu ambiente, inicialmente no formato padrão .dwg, para o formato .fbx, de forma simples.

O formato de arquivo FBX (do inglês, Filmbox), por sua vez, é um *framework* aberto para transferência de dados de objetos 3D, o qual cria um alto nível de interoperabilidade entre os programas da AutoDesk. Diversas outras plataformas de desenvolvimento 3D também suportam a importação de arquivos FBX, tais como Blender, Unreal e Unity, o que permite modularizar e otimizar o desenvolvimento de soluções 3D, aproveitando as funcionalidades específicas de cada ferramenta.

2.4.2 Unity

O Unity é um motor de jogos multiplataforma desenvolvido pela empresa Unity Technologies, o qual pode ser utilizado para criar jogos 2D, 3D, em RV ou RA, bem como simulações e outras experiências.

Figura 5 – Interior desenvolvido no ambiente do Unity.



Fonte: Unity.

O motor oferece aos usuários uma API (do inglês, *Application Programming Interface* de desenvolvimento de códigos na linguagem de programação C#, assim como a funcionalidade de arrastar e soltar, o que otimiza a etapa de construção e testes da lógica que se deseja implementar. É possível integrar SDKs (do inglês, *Software Development Kits*) e *plugins* desenvolvidos por terceiros, o que possibilita a sua utilização para uma grande variedade de aplicações.

O Unity Editor, o qual é o ambiente de desenvolvimento das soluções, possui suporte para as plataformas Windows, macOS e Linux, enquanto que a aplicação final pode ser construída para mais de 25 plataformas diferentes, tais como Android, Windows e Linux, nas categorias *mobile*, *desktop* e *console*, dentre outras. Nesse ambiente é possível utilizar comandos para a criação de objetos 2D e 3D, além da integração de modelos físicos a estes, assim como a inserção de componentes de interface com o usuário.

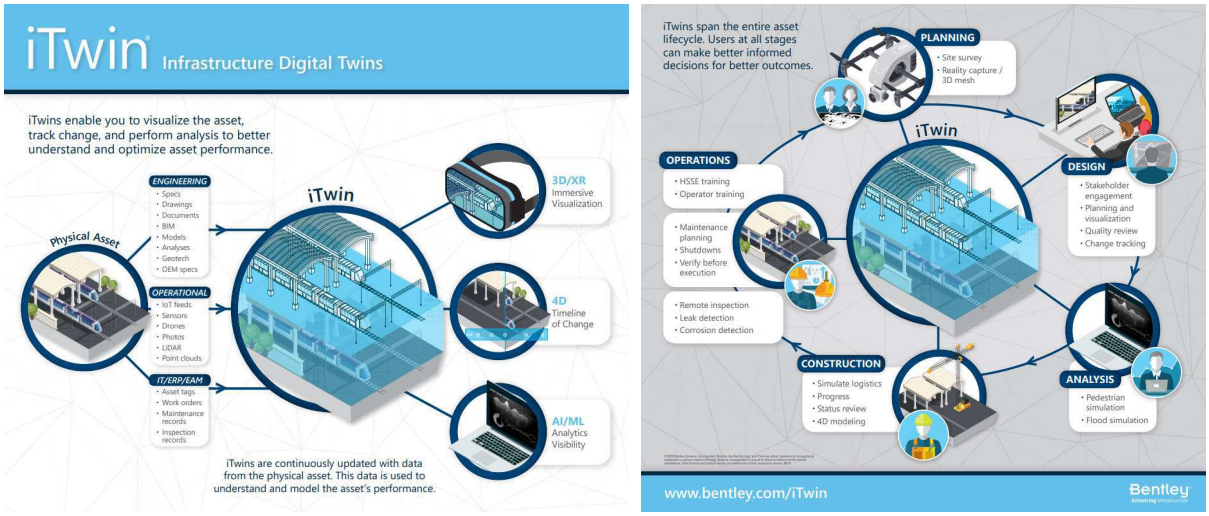
No ano de 2018, estima-se que o Unity tenha sido utilizado na criação de aproximadamente metade dos jogos *mobile* no mercado e 60% dos conteúdos de RA e RV, incluindo aproximadamente 90% das soluções nas plataformas emergentes de RA, tais como o Microsoft Hololens (MEDIA, 2017).

2.5 Ferramentas alternativas

2.5.1 iTwin

O *software* iTwin funciona a partir de imagens de vídeo capturadas por drones, onde são criados modelos em 3D da região de interesse. Feito isso, há uma identificação automática de plantas, estruturas e maquinário, aos quais podem ser associadas informações adquiridas em tempo real, dos sistemas físicos.

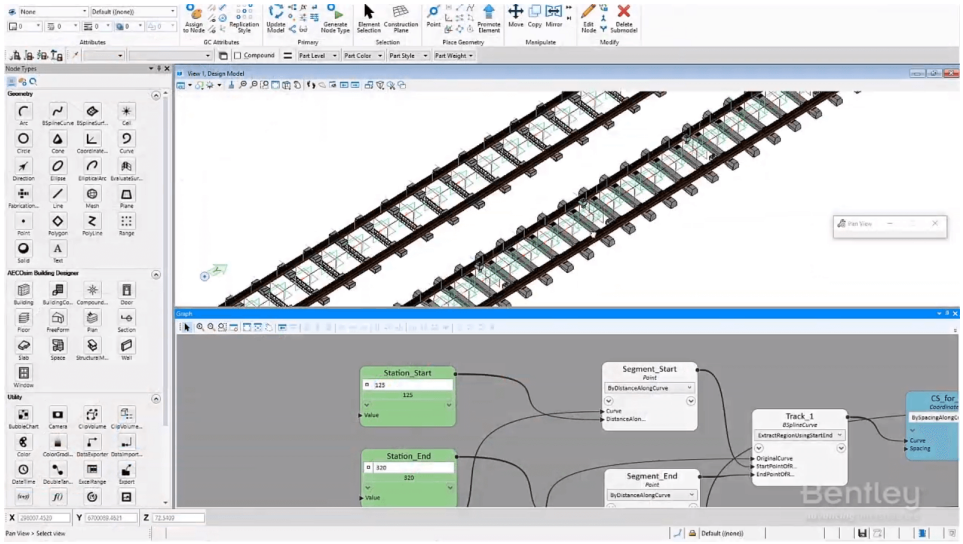
Figura 6 – Resumo de funcionamento do *software* iTwin.



Fonte: Bentley Systems.

É possível ainda, diretamente em seu ambiente, construir modelos virtuais e realizar simulações, como visto na Figura 7.

Figura 7 – Construção de modelos virtuais no iTwin.



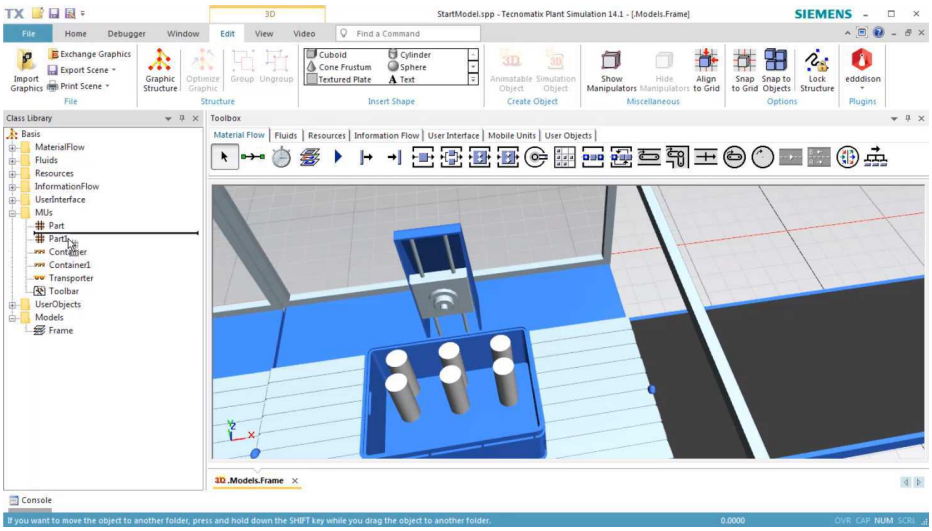
Fonte: Bentley Systems.

2.5.2 Tecnomatix Plant Simulation

No *software* Tecnomatix Plant Simulation, por meio de ferramentas de visualização gráfica e análise de dados, além da utilização de algoritmos genéticos e ferramentas de teste, é possível avaliar o comportamento dos sistemas de produção em diversos cenários.

É ainda possível realizar análises estatísticas relacionadas ao funcionamento dos objetos inseridos no ambiente, tanto de forma específica, em elementos individuais, quanto de forma geral, em toda a planta. Adicionalmente, o *software* permite conectar o modelo virtual da planta tanto a um CLP virtual quanto a um CLP físico de uma planta real com o objetivo de simular uma produção real.

Figura 8 – Configuração de objetos 3D no Tecnomatix Plant Simulation.



Fonte: Siemens.

Figura 9 – Realização de simulação no Tecnomatix Plant Simulation.

Fonte: Siemens.

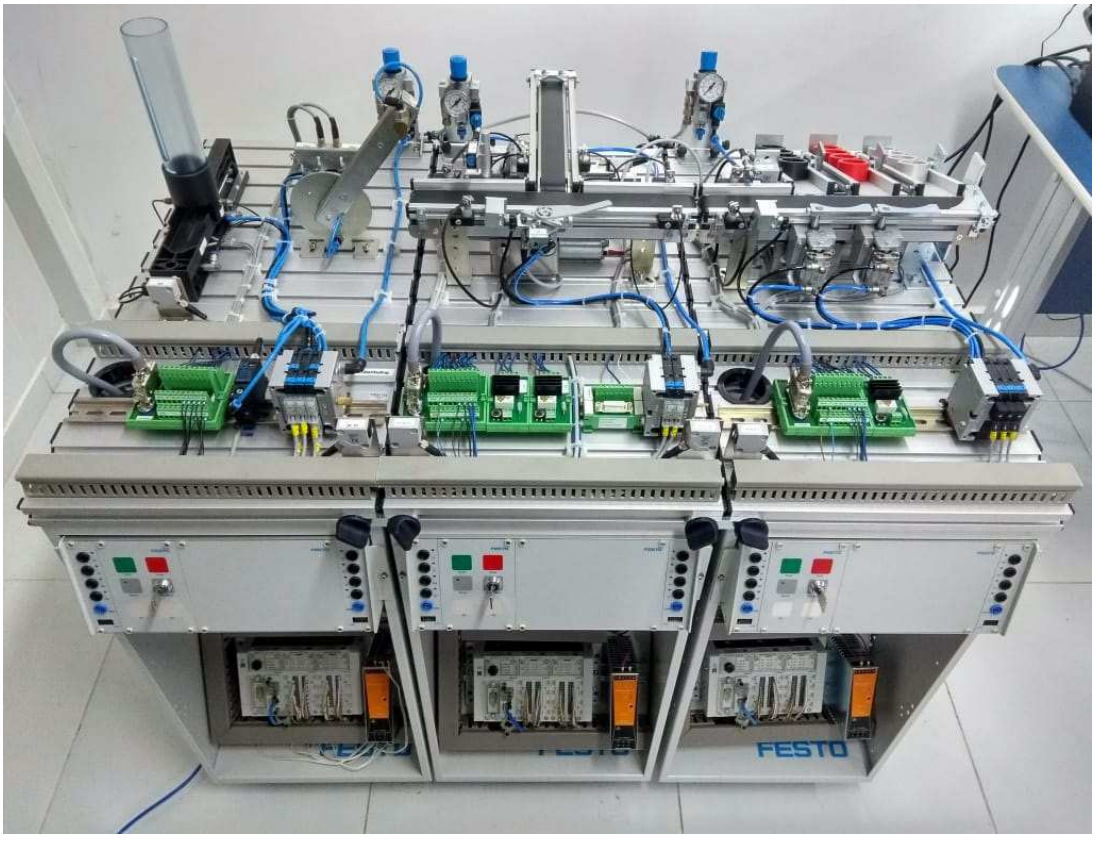
3 Desenvolvimento do gêmeo digital

O presente capítulo trata dos passos relativos ao desenvolvimento do gêmeo digital do MPS da Festo. Mais especificamente, a modelagem 3D dos componentes do sistema no *software* AutoCAD, a exportação desse modelo para o motor de jogos Unity, o desenvolvimento da interface com o usuário, a utilização de recursos de realidade aumentada e, finalmente, a implementação de um cliente OPC UA integrado à aplicação.

3.1 Modelagem 3D

Como mencionado anteriormente, escolheu-se o *software* AutoCAD para esta etapa de modelagem 3D do MPS (Figura 10), em detrimento de outros disponíveis gratuitamente pela familiaridade com sua interface, comandos facilmente memorizáveis e, principalmente, por permitir a exportação direta dos modelos no formato .dwg para o formato .fbx, o qual pode, por sua vez, ser importado diretamente no Unity3D.

Figura 10 – MPS da Festo disponível no LIEC.

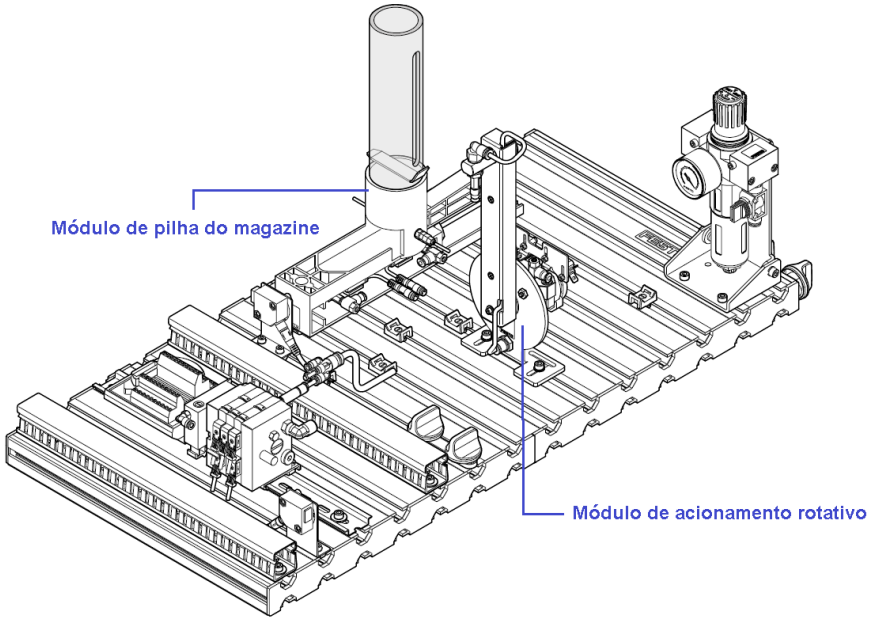


Fonte: Autoria própria.

3.1.1 Estação de Distribuição

A estação de distribuição é a estação modular inicial do MPS, e é responsável pela distribuição das peças para as demais estações.

Figura 11 – Esquema da estação de distribuição.



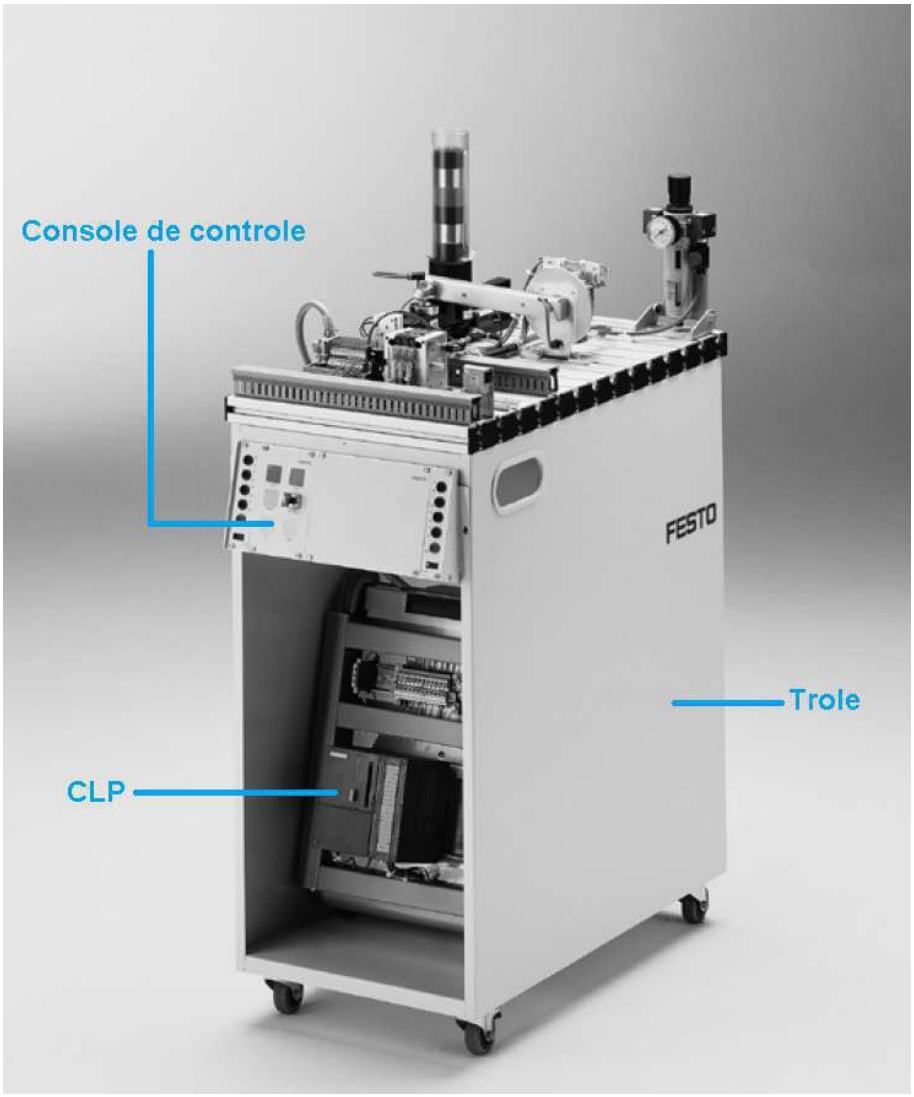
Fonte: Adaptado de (EBEL; PANY, 2006a).

As estações são constituídas de módulos menores responsáveis por funções específicas, as quais serão mais detalhadas na seção 3.2. A estação de distribuição possui, essencialmente, o módulo de pilha do magazine e o módulo de acionamento rotativo, como seus módulos funcionais, isto é, que são responsáveis diretamente por alcançar o objetivo da estação.

Levando em consideração os aspectos de tempo despendido e simplicidade da solução, decidiu-se por modelar no *software* AutoCAD apenas os módulos funcionais, associados à placa de perfil, removendo-se os demais componentes, referentes ao sistema pneumático e elétrico, além do console de controle, o CLP e o trole (Figura 12).

Como mencionado anteriormente, buscou-se desenvolver os sólidos em 3D dos módulos funcionais da estação de distribuição de acordo com estimações a partir das fontes referenciadas e de fotos, tomando o cuidado de respeitar ao máximo as proporções.

Figura 12 – Estação de distribuição isolada.



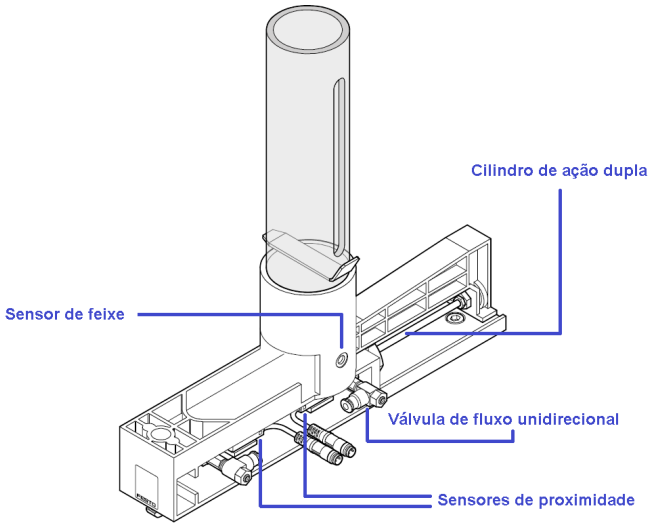
Fonte: Adaptado de (EBEL; PANY, 2006a).

3.1.1.1 Módulo de pilha do magazine

Os componentes do módulo de pilha do magazine e suas funções resumidas são os seguintes:

- Cilindro de ação dupla: empurra a peça do cilindro do magazine para o ponto de transferência, no qual o módulo de troca irá levá-la;
- Sensor de feixe: monitora a disponibilidade de peças no cilindro do magazine;
- Sensores de proximidade: medem a distância com relação ao cilindro de ação dupla;
- Válvula de fluxo unidirecional: permite ajustar a velocidade do cilindro de ação dupla;

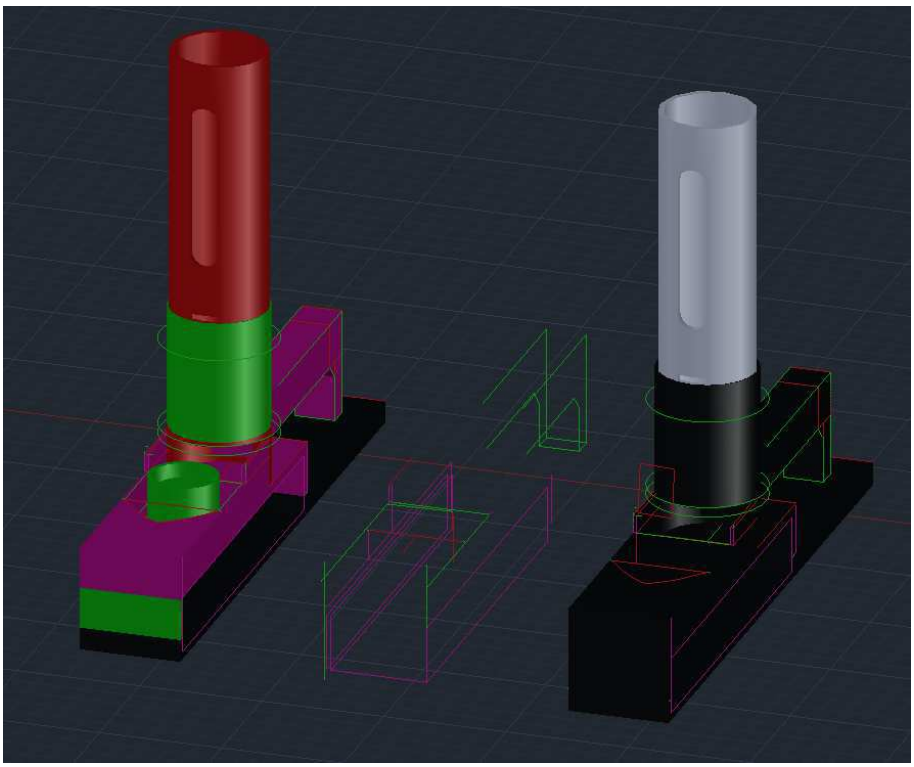
Figura 13 – Módulo de pilha do magazine.



Fonte: Adaptado de (EBEL; PANY, 2006a).

Optou-se por modelar apenas o magazine, o cilindro do magazine e o cilindro de ação dupla, já que os sensores estão em posições discretas e podem ser abstraídos. Os esboços iniciais, assim como o modelo 3D final do módulo de pilha do magazine são apresentados na Figura 14.

Figura 14 – Modelo 3D do módulo de pilha do magazine.



Fonte: Autoria própria.

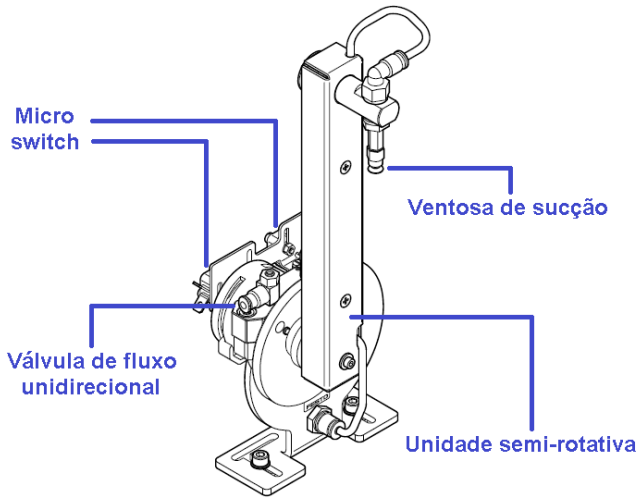
As linhas de esboço inicial foram feitas com o comando *line*, escolhendo-se cores que contrastassem entre si para melhor identificação das regiões e partes de cada componente do módulo. Os sólidos foram construídos por meio do comando *extrude*, tendo sido utilizado também o comando *subtract* para as formas cilíndricas com furos.

3.1.1.2 Módulo de acionamento rotativo

Os componentes do módulo de acionamento rotativo e suas funções resumidas são os seguintes:

- Micro *switch*: mede a posição da unidade semi-rotativa;
- Unidade semi-rotativa: atuador pneumático de ação dupla que pode rotacionar entre 90º a 270º, com válvulas de controle de fluxo;
- Válvula de fluxo unidirecional: permite ajustar a velocidade da unidade semi-rotativa;
- Ventosa de sucção: permite que uma peça fique presa à unidade semi-rotativa e, assim, levada à outra estação.

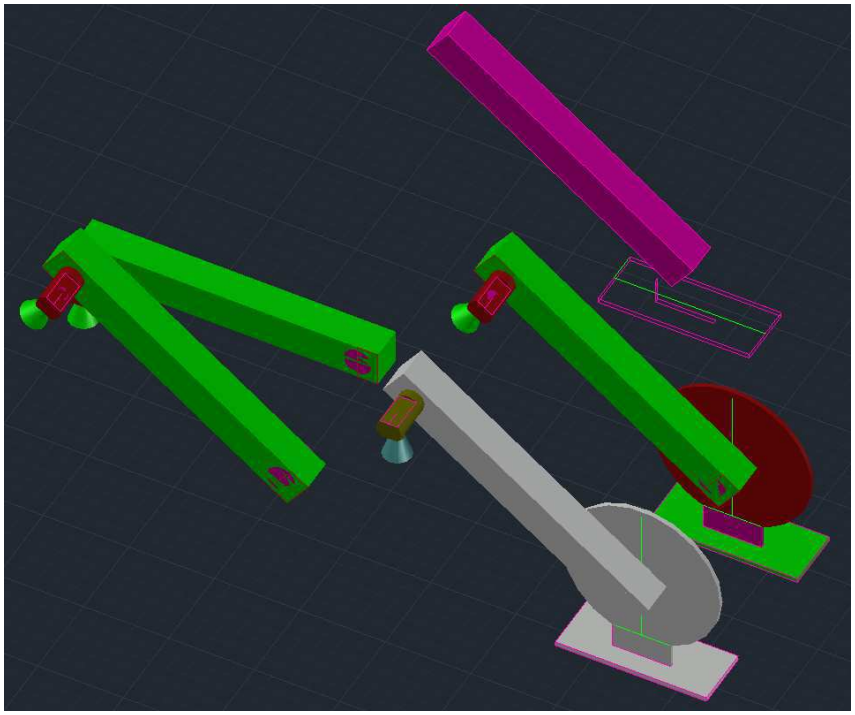
Figura 15 – Módulo de acionamento rotativo.



Fonte: Adaptado de (EBEL; PANY, 2006a).

Optou-se por modelar apenas a unidade semi-rotativa e a ventosa de sucção, omitindo-se o micro *switch* e a válvula de fluxo unidirecional, pelos mesmos motivos do módulo de pilha do magazine. Os esboços iniciais referentes a este módulo no AutoCAD são apresentados na Figura 16.

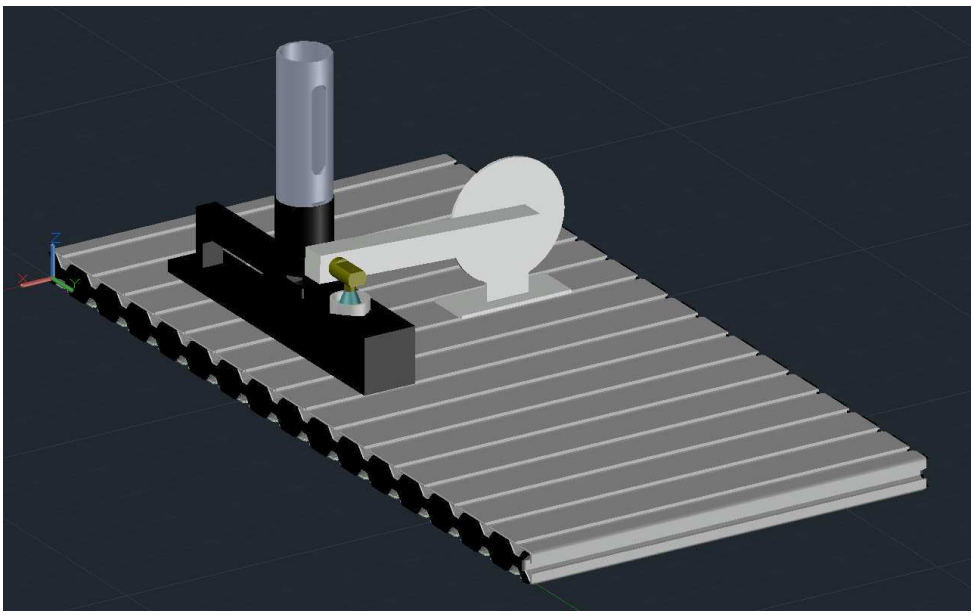
Figura 16 – Modelo 3D do módulo de acionamento rotativo.



Fonte: Autoria própria.

Finalmente, tomou-se o cuidado de unir os módulos respeitando a simetria e o posicionamento destes com relação à estação de distribuição real tomando-se como referência a peça na posição final do magazine, onde é coletada pela unidade semi-rotativa. O modelo 3D final da estação de distribuição é apresentado na Figura 17.

Figura 17 – Modelo 3D da estação de distribuição no AutoCAD.

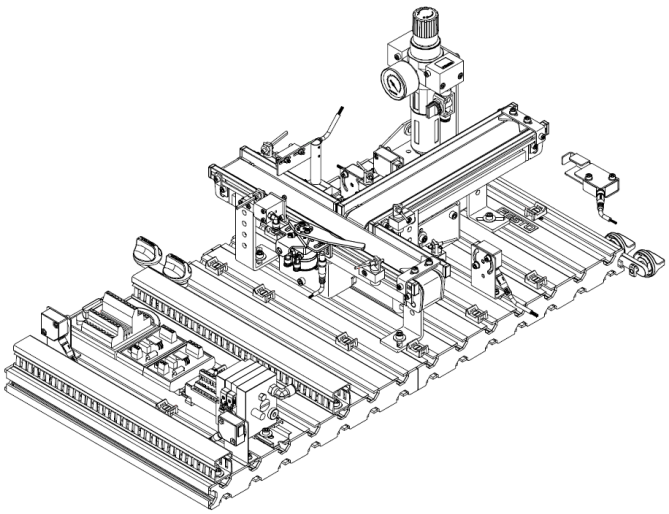


Fonte: Autoria própria.

3.1.2 Estação de Separação

A estação de separação é a segunda estação modular na sequência do MPS presente no laboratório, e é responsável pela separação das peças com furo orientado para cima, daquelas com furo orientado para baixo. As peças são, então, desviadas para a segunda esteira, a depender deste parâmetro.

Figura 18 – Esquema da estação de separação.



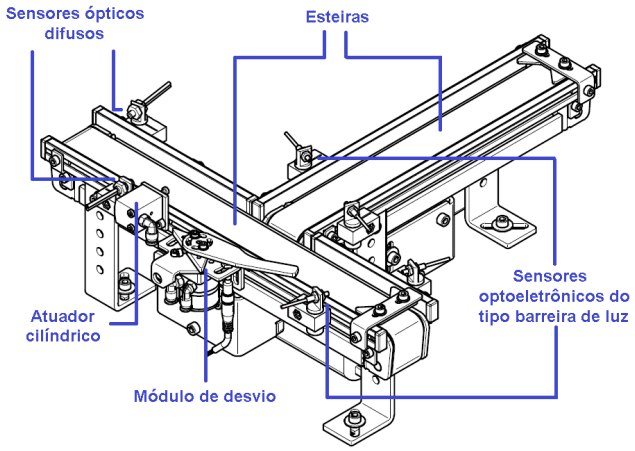
Fonte: Adaptado de (EBEL; PANY, 2006b).

Os componentes da estação de separação e suas funções resumidas são os seguintes:

- Atuador cilíndrico: interrompe o movimento da peça por meio de bloqueio;
- Esteiras: movimentam as peças em apenas um sentido;
- Módulo de desvio: desvia a peça da esteira inicial para a segunda a depender da orientação da peça;
- Sensor de distância: determina a orientação da peça a partir da luz refletida na cavidade.
- Sensores optoeletrônicos do tipo barreira de luz: detectam a passagem das peças;
- Sensores ópticos difusos: detectam a presença de uma peça na esteira.

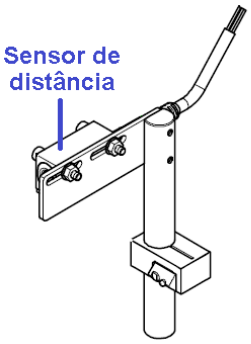
Optou-se por modelar todos os componentes do módulo de separação, com exceção das partes não funcionais do módulo de desvio e das esteiras.

Figura 19 – Módulo de separação.



Fonte: Adaptado de (EBEL; PANY, 2006b).

Figura 20 – Sensor de distância.

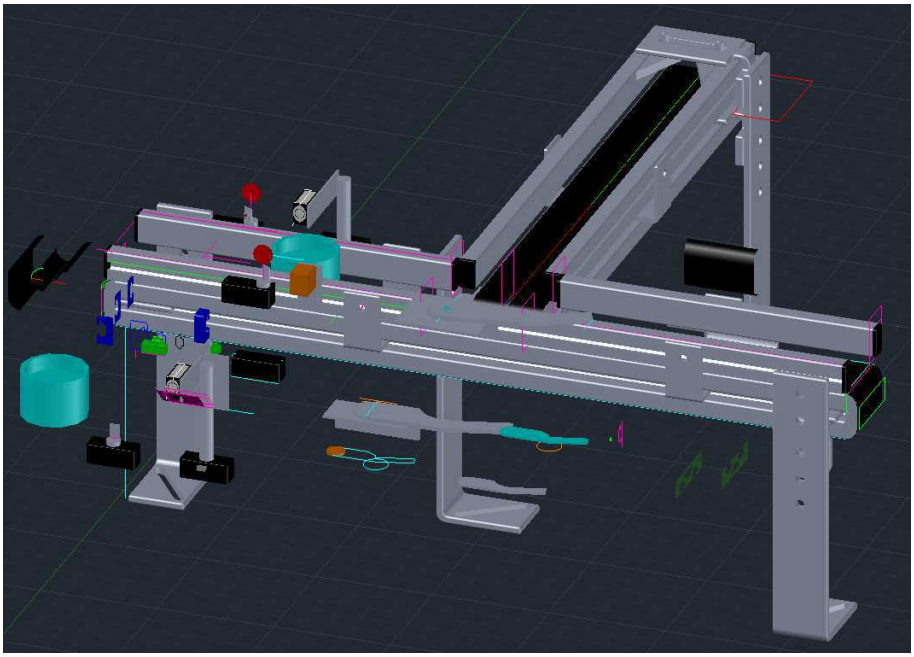


Fonte: Adaptado de (EBEL; PANY, 2006b).

De maneira similar à estação de distribuição, tomou-se o cuidado de atentar para a proporção e simetria dos componentes, tomando como referência a placa de perfil para dimensionar as esteiras e modelando as demais partes a partir das dimensões destas. Como esta estação possui uma quantidade significativamente maior de módulos e componentes, os detalhes relativos às etapas de modelagem 3D no *software* foram omitidos. O desenvolvimento inicial do modelo, assim como a versão final são apresentados nas Figuras 21 e 22, respectivamente.

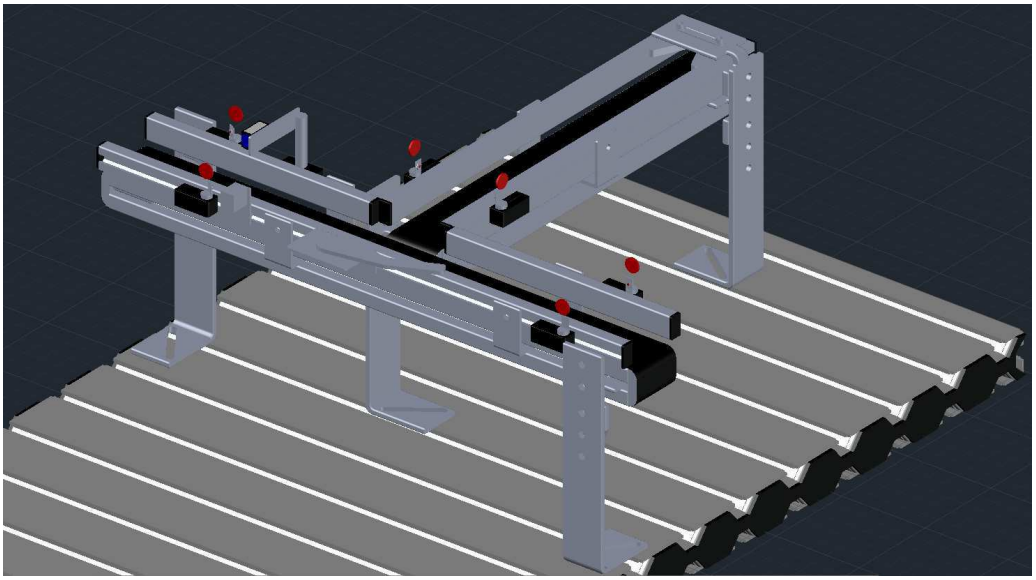
Os cilindros de cor vermelha logo acima dos sensores ópticos não refletem nenhum componente da estação e foram adicionados para ajudar os usuários na compreensão do funcionamento do sistema. Esta etapa será melhor detalhada na seção 3.2.

Figura 21 – Desenvolvimento do modelo 3D da estação de separação.



Fonte: Autoria própria.

Figura 22 – Modelo 3D final da estação de separação.

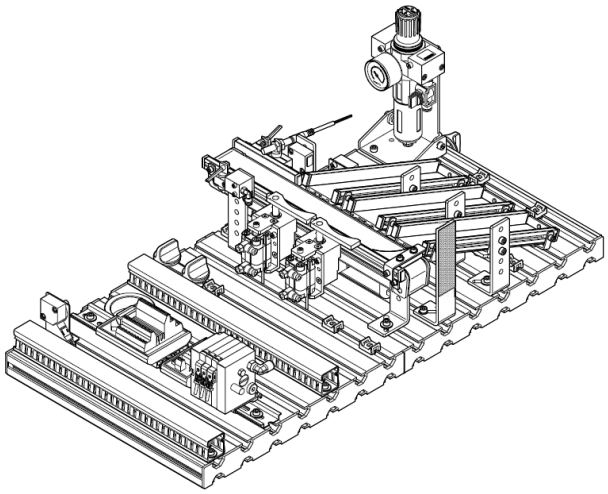


Fonte: Autoria própria.

3.1.3 Estação de Classificação

A estação de classificação é a terceira e última estação modular na sequência do MPS presente no laboratório, e é responsável por organizar as peças de acordo com seu material: metálico ou de plástico, e sua cor: vermelho ou preto. Na Figura 23 apresenta-se um esquema desta estação.

Figura 23 – Esquema da estação de classificação.

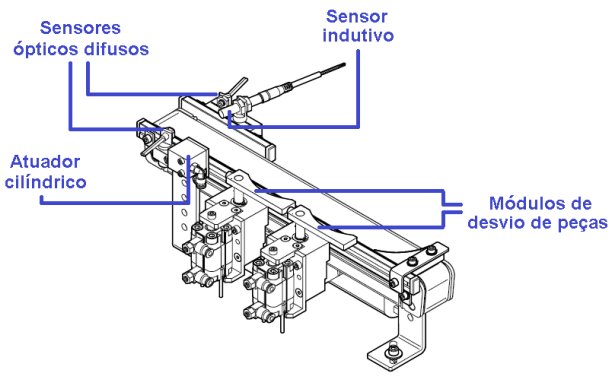


Fonte: Adaptado de (EBEL; PANY, 2006c).

Os componentes da estação de separação e suas funções resumidas são os seguintes:

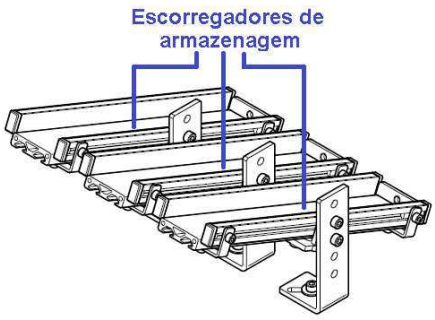
- Atuador cilíndrico: interrompe o movimento da peça por meio de bloqueio;
- Esteira: movimenta as peças em apenas um sentido;
- Módulos de desvio: desviam a peça da esteira para a armazenagem a depender dos sinais do sensor óptico e do sensor indutivo;
- Sensores ópticos difusos: detectam a presença e a cor de peças na esteira;
- Sensor indutivo: detecta a presença de peças metálicas.

Figura 24 – Módulo de detecção do tipo de peça.



Fonte: Adaptado de (EBEL; PANY, 2006c).

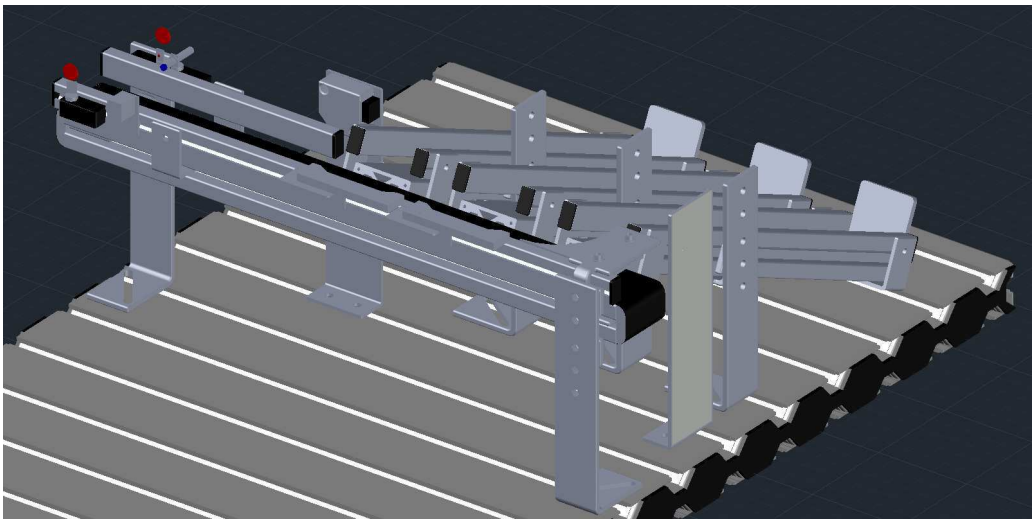
Figura 25 – Módulo escorregador.



Fonte: Adaptado de (EBEL; PANY, 2006c).

Optou-se por modelar todos os componentes do módulo de classificação, com exceção de algumas partes não funcionais dos módulos de desvio e da esteira. Foi possível aproveitar grande parte dos componentes previamente modelados para a estação de separação, de forma que o passo de esboço inicial das partes restantes foi omitido. O modelo 3D desenvolvido para a estação de classificação é apresentado na Figura 26.

Figura 26 – Modelo 3D da estação de classificação.



Fonte: Autoria própria.

3.2 Replicação da Dinâmica do Sistema

Concluída a etapa de modelagem em 3D dos componentes dos sistemas, estes foram exportados para a *engine* Unity 3D para o desenvolvimento da replicação da dinâmica do sistema real no sistema virtual construído.

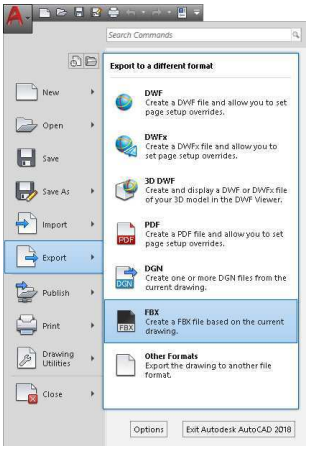
A *engine* Unity3D suporta a importação de malhas e animações de dois tipos diferentes de arquivos: formatos de arquivo 3D exportados, como .fbx ou .obj, ou arquivos de aplicativos proprietários 3D ou DCC (*Digital Content Creation*) (UNITY, 2018).

Considerando o fato de que o *software* AutoCAD, em sua versão do ano de 2018, permite a exportação dos modelos construídos para o formato .fbx, mesmo na versão para estudantes (AUTOCAD,), optou-se por este método para exportar os modelos do AutoCAD para o Unity3D. Além disto, este formato ocupa, na memória, cerca de 10% de seu equivalente no formato .dwg, o que contribui significativamente para a performance da aplicação construída.

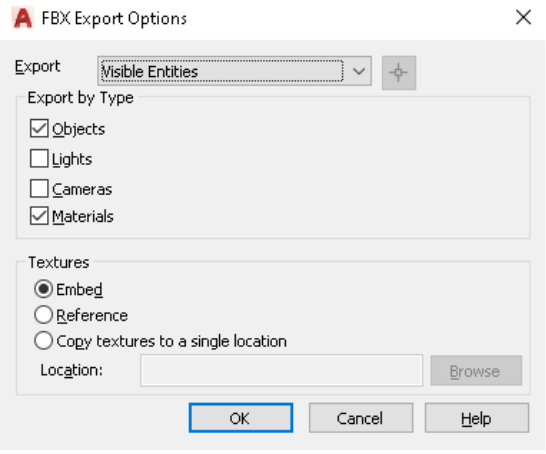
Para exportar os modelos em 3D construídos no AutoCAD, basta clicar no menu Aplicativo e selecionar a opção *Export* → FBX, como está apresentado na figura 27a. Feito isto, o usuário deve escolher o nome do arquivo e o local na memória em que deseja salvá-lo. Uma caixa de diálogo será exibida, como se pode ver na figura 27b, para que o usuário escolha se deseja exportar as entidades visíveis ou apenas as selecionadas, os tipos que deseja exportar e o modo de exportação de possíveis texturas inseridas nos modelos.

Figura 27 – Exportação dos modelos construídos em .dwg para .fbx.

(a) Exportação FBX.



(b) Opções de exportação FBX.



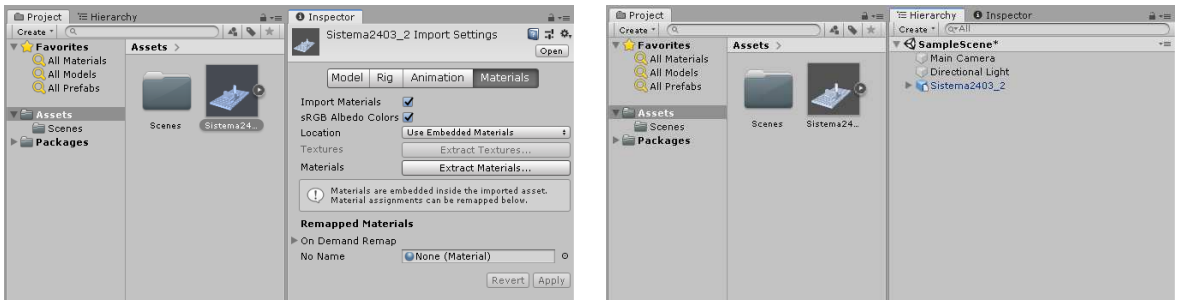
Fonte: Autoria própria.

Para fins de simplicidade, decidiu-se por não exportar as luzes e as câmeras, inserindo-as apenas no próprio Unity3D, e, embora tenha-se escolhido por exportar os materiais, estes também foram alterados posteriormente no Unity3D. Optou-se também pela opção *Embed*, com relação às texturas, embora não se tenha utilizado delas para os modelos desenvolvidos.

Como mencionado anteriormente, o Unity3D é capaz de ler os arquivos no formato .fbx diretamente, de tal forma que se pode inserir o arquivo no ambiente da *engine* arrastando-o e soltando-o na pasta *Assets*, como visto na figura 28a e, após isto, arrastando-o e soltando-o dentro da cena, na aba *Hierarchy*, como visto na figura 28b.

Figura 28 – Importação do arquivo .fbx no ambiente do Unity3D.

(a) Importação FBX. (b) Inserção do objeto na cena.

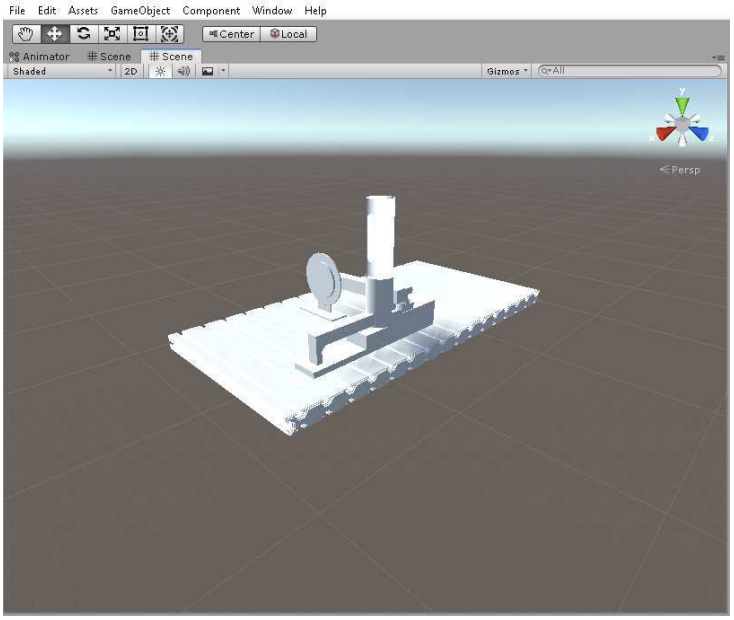


Fonte: Autoria própria.

É possível perceber que o ícone do objeto inserido difere dos ícones dos demais objetos, e isso se deve ao fato de que o arquivo .fbx inserido no ambiente é importado como um *Prefab*. Um *Prefab* no Unity3D é, essencialmente, um objeto complexo, que agrega todos os seus componentes, valores das propriedades e objetos filhos, de forma hierarquizada, em um ativo reutilizável dentro do ambiente. O *Prefab* age, portanto, como um template, a partir do qual se pode criar novas instâncias do mesmo ativo na cena.

O resultado da inserção pode ser visto na figura 31, onde se identifica que as cores atribuídas aos objetos no *software* AutoCAD não são importadas como materiais no Unity3D, pois são parâmetros distintos. O material do *Prefab* no Unity3D é o mesmo material definido no AutoCAD, o qual é um material padrão.

Figura 29 – Objeto .fbx importado no Unity3D.



Fonte: Autoria própria.

3.2.1 Estação de Distribuição

O funcionamento da estação de distribuição com relação aos seus componentes modelados é detalhado a seguir.

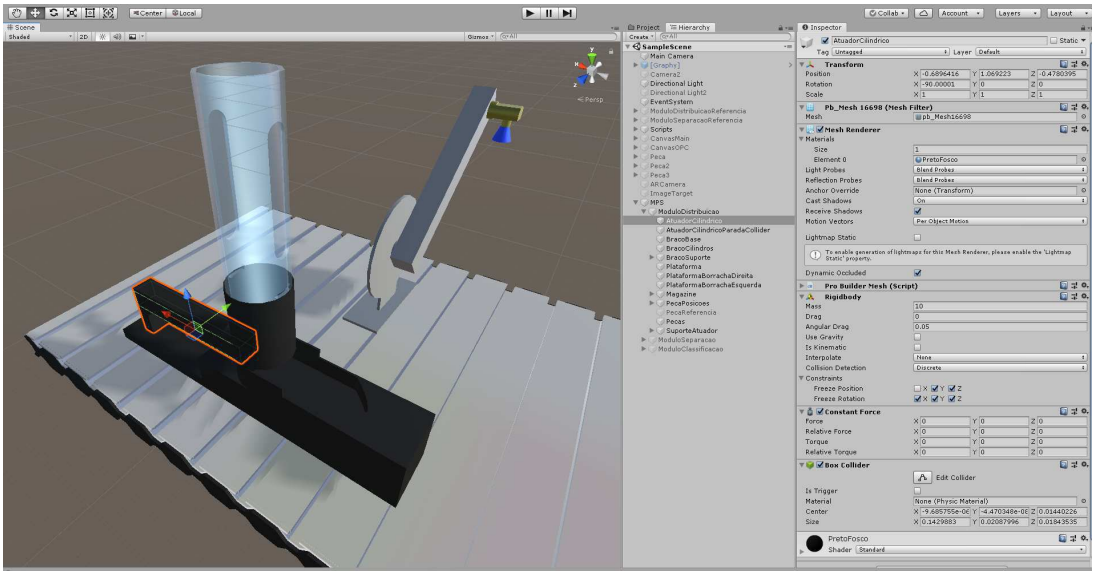
Inicialmente, as peças devem ser inseridas manualmente no cilindro do magazine, o qual possui capacidade máxima de 8 peças. O sensor de presença de peças no magazine deve indicar se há, ou não, peças no cilindro do magazine. O atuador cilíndrico, por sua vez, quando acionado, desloca a peça no magazine de sua posição inicial para a posição final onde deverá ser coletada pelo módulo semi-rotativo.

A unidade semi-rotativa deve ser rotacionada, a partir de sua origem, de um ângulo total capaz de fazer tocar a ventosa de sucção na peça. A ventosa, por sua vez, deve manter um ângulo de 0º com relação ao plano da peça, de tal forma que o vácuo possa se estabelecer plenamente e a peça possa se conectar com estabilidade à ventosa.

A peça, após o estabelecimento do vácuo, pode ser transportada de sua posição de coleta, no magazine, para a sua posição de entrega, na esteira da estação de distribuição. Tal transporte se dá por meio, inicialmente, da rotação da unidade semi-rotativa à seu ângulo original e, posteriormente, pelo desligamento do vácuo, o qual permite que a peça se desconecte e caia, por gravidade, na esteira da estação de distribuição.

A partir da descrição do funcionamento da estação, é possível identificar que a dinâmica a ser replicada consiste em implementar, no Unity3D, a translação de objetos e a rotação, em um único plano. A fim de realizar tal implementação, alguns passos são requeridos, os quais serão detalhados a seguir.

Figura 30 – Estação de distribuição - Atuador cilíndrico.



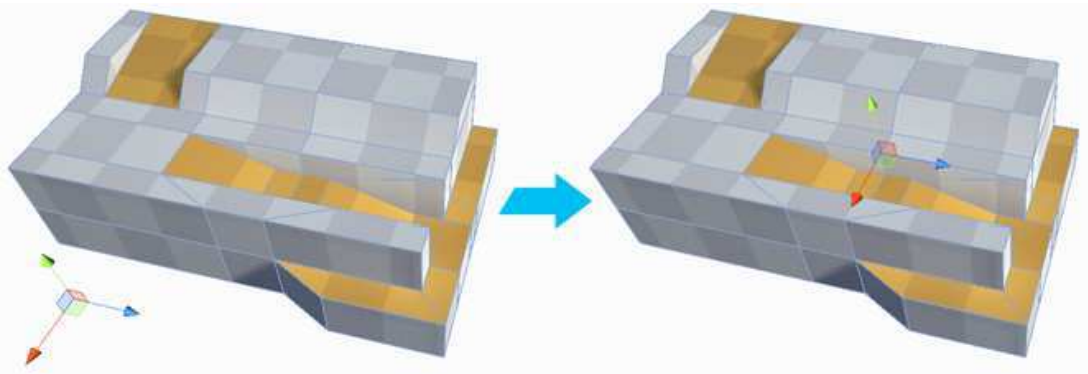
Fonte: Autoria própria.

Inicialmente, realizou-se o *unpacking* do Prefab, clicando com o botão direito no objeto na hierarquia e selecionando esta opção, a fim de tratar os componentes do objeto como *Gameobjects* regulares dentro da cena. Estes, por sua vez, são os objetos fundamentais do Unity e atuam como contêineres para os componentes disponibilizados no ambiente (materiais, colisores, propriedades físicas e etc.). Por meio do agregamento de tais componentes, é possível implementar a funcionalidade real.

Considerando o fato de que, o atuador cilíndrico possui uma dinâmica de duração muito curta, sendo sua ação quase instantânea, verificou-se que a implementação desta no ambiente virtual requer grande precisão, para que seja realizada de maneira realista, sem distorções gráficas.

A fim de alcançar tal objetivo, foi adicionado ao programa o *plugin* ProBuilder. Este *plugin* possui, dentre outras funcionalidades, a opção de centralizar o pivô do objeto no seu centro. A opção *Center Pivot* move o ponto pivô da Malha para o centro dos limites do objeto. Isso se faz necessário pelo fato de que os objetos, por padrão, se movimentam com base no seu pivô, e este pode não estar alinhado com o centro do objeto, como é o caso de componentes de *Prefabs*.

Figura 31 – Opção *Center Pivot* do *plugin* Probuilder.

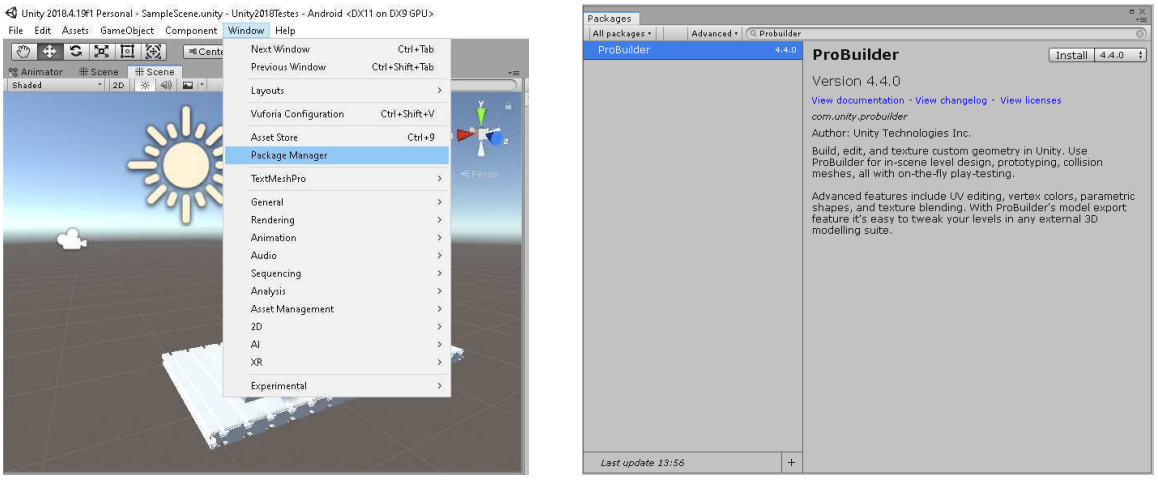


Fonte: Unity.

Para adicionar o *plugin* Probuilder ao Unity3D basta abrir o gerenciador de pacotes clicando em *Window* → *Package Manager*, inserir ProBuilder na caixa de pesquisa, selecionar a opção correspondente e clicar no botão *Install*. Estes passos são apresentados nas figuras 32a e 32b.

Figura 32 – Instalando o *plugin* ProBuilder Unity3D.

(a) Gerenciador de pacotes. (b) Instalação do *plugin*.

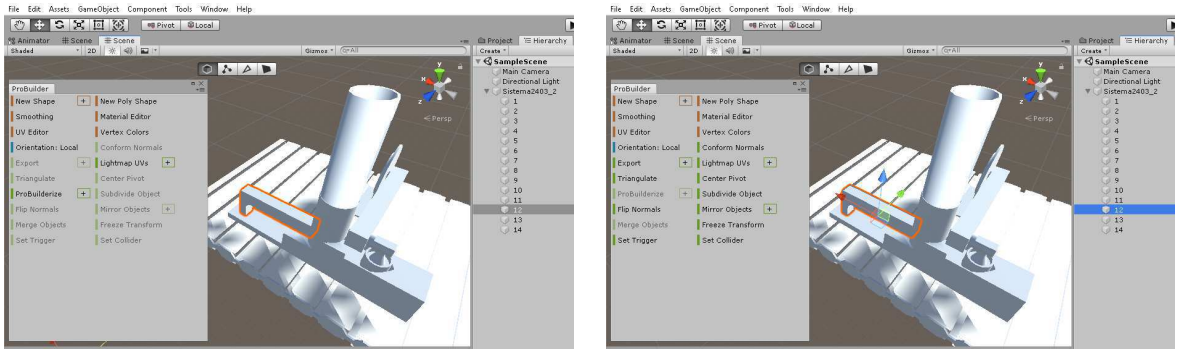


Fonte: Autoria própria.

Para aplicar a funcão *Center Pivot* deve-se clicar na nova aba disponibilizada Tools → ProBuilder → ProBuilder Window. Com a janela aberta, ao clicar no objeto desejado, a opção ProBuilderize se tornará disponível. A função ProBuilderize converte os objetos selecionados em objetos editáveis pelo ProBuilder. Após isto, deve-se clicar na função *Center Pivot*, que estará disponível, como visto nas figuras 33a e 33b.

Figura 33 – Aplicando a função *Center Pivot*.

(a) Função ProBuilderize. (b) Função *Center Pivot*.



Fonte: Autoria própria.

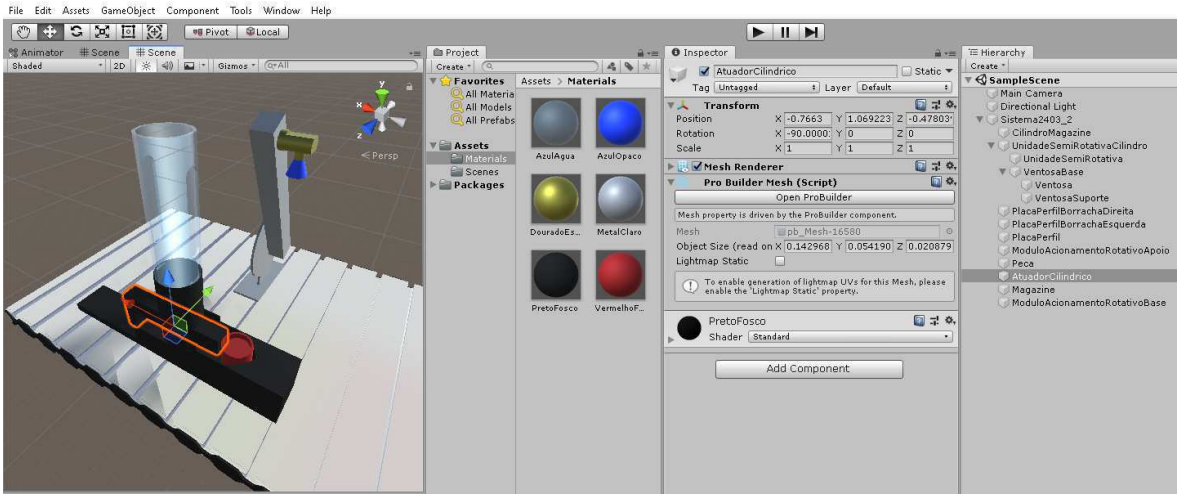
Realizou-se os mesmos passos para o módulo de acionamento rotativo, mais especificamente seus componentes: unidade semi-rotativa e ventosa, organizando-os de forma hierárquica, o que permite a rotação conjunta a partir de uma mesma referência. Além disso, nomeou-se os componentes da estação adicionando materiais a estes de forma a facilitar sua visualização e identificação a partir desta etapa.

Buscou-se utilizar materiais simples, mas que possibilitassem uma visualização realista dos componentes da estação. Os seguintes materiais foram utilizados:

- Azul água: azul translúcido, aplicado ao cilindro do magazine;
- Azul opaco: azul opaco, aplicado à ventosa;
- Dourado escuro: amarelo metálico escuro, aplicado à base da ventosa;
- Metal claro: cinza metálico, aplicado à placa de perfil;
- Preto fosco: preto opaco, aplicado ao atuador cilíndrico, borrachas laterais conectadas à placa de perfil e magazine;
- Vermelho fosco: vermelho opaco, aplicado à peça.

Movimentou-se o atuador cilíndrico manualmente no eixo x de forma a estimar sua posição final desejada no magazine, tendo a peça como referência. De forma a facilitar a visualização, também rotacionou-se a unidade semi-rotativa, compensando esta rotação na ventosa, o que pode ser visto na figura 34.

Figura 34 – Replicação da dinâmica do atuador cilíndrico.



Fonte: Autoria própria.

Tendo-se estas referências de posição, é possível, então, iniciar o desenvolvimento do código que irá realizar a movimentação do componente durante a execução do aplicativo. Para isto, basta clicar com o botão direito dentro da aba *Project* → *Create* → C# *Script*. Será criado um arquivo texto com um código base em C# a partir do qual o desenvolvedor pode implementar as funcionalidades desejadas. Este pode ser desenvolvido em qualquer IDE de escolha, sendo compilado e atualizado automaticamente no ambiente do Unity.

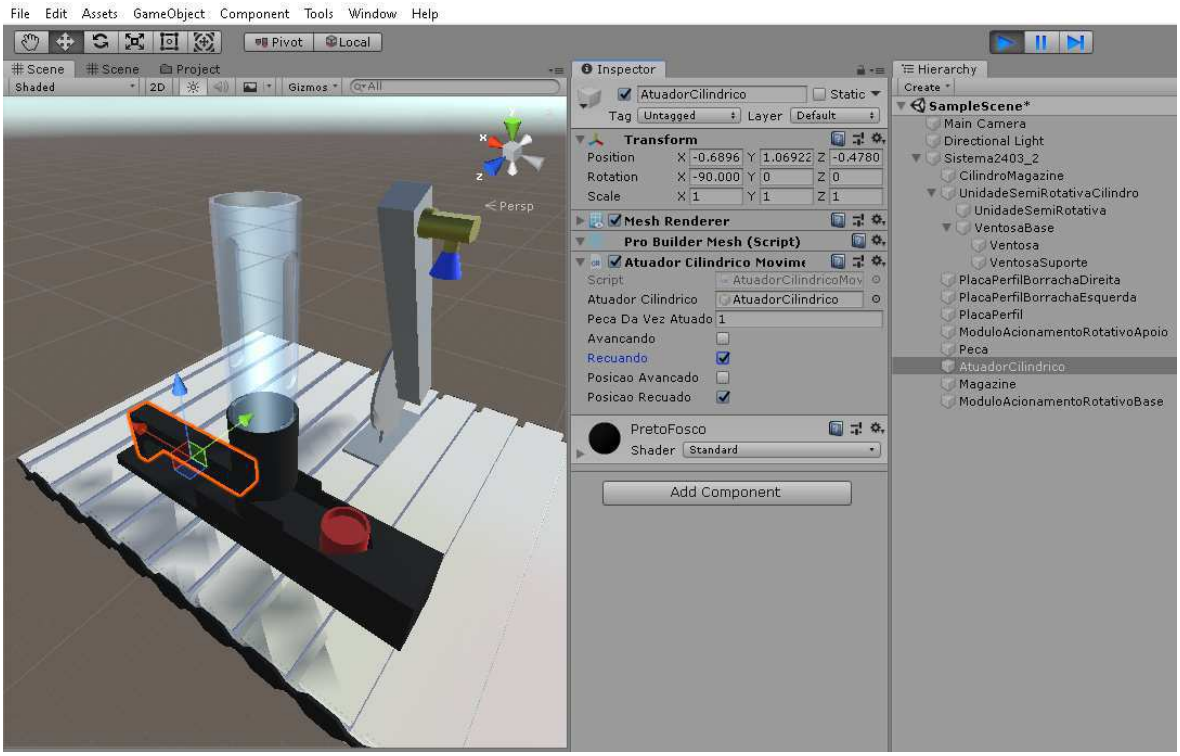
A dinâmica do atuador cilíndrico consiste, portanto, em ser movimentado no eixo x, a partir de comandos individuais de acionamento e de desligamento. Para isto, no Unity3D, é possível utilizar-se de diversos métodos para movimentar objetos, porém, optou-se por utilizar o incremento (ou decremento, no caso de recuo do atuador) da propriedade transform.position do *gameobject*.

$$AtuadorCilindrico.transform.position+ = transform.right * distanciafloat;$$

A partir deste método, o atuador cilíndrico será movimentado no eixo x, a uma taxa correspondente à variável *distanciafloat*, a qual refere-se à distância entre a posição inicial e a posição final do atuador. A função *Update*, na qual o código inserido, é chamada em todo o frame da execução, de tal forma que o resultado final é um movimento contínuo e fluido.

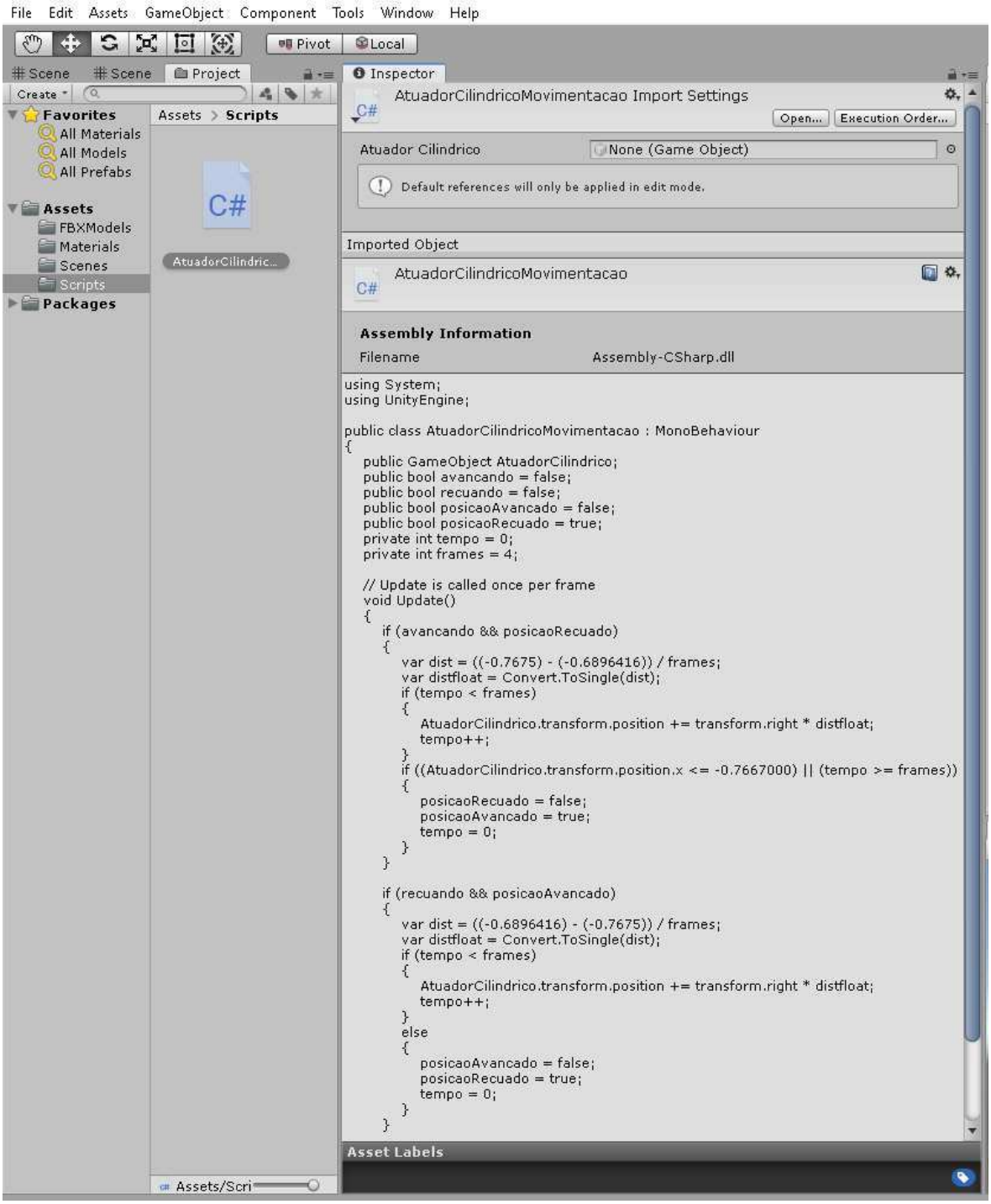
Terminada a escrita do código em C#, para permitir a sua atuação na cena, basta arrastar e soltar os *gameobjects* definidos como propriedades da classe, na região correspondente do editor do script. Após isto, basta arrastar e soltar o script na lista de componentes de um *gameobject* da cena, na aba *Inspector*, para que o código seja executado quando iniciada a aplicação, como pode ser visto na figura 35.

Figura 35 – Implementação da dinâmica do atuador cilíndrico.



Fonte: Autoria própria.

Figura 36 – Código inicial da dinâmica do atuador cilíndrico.



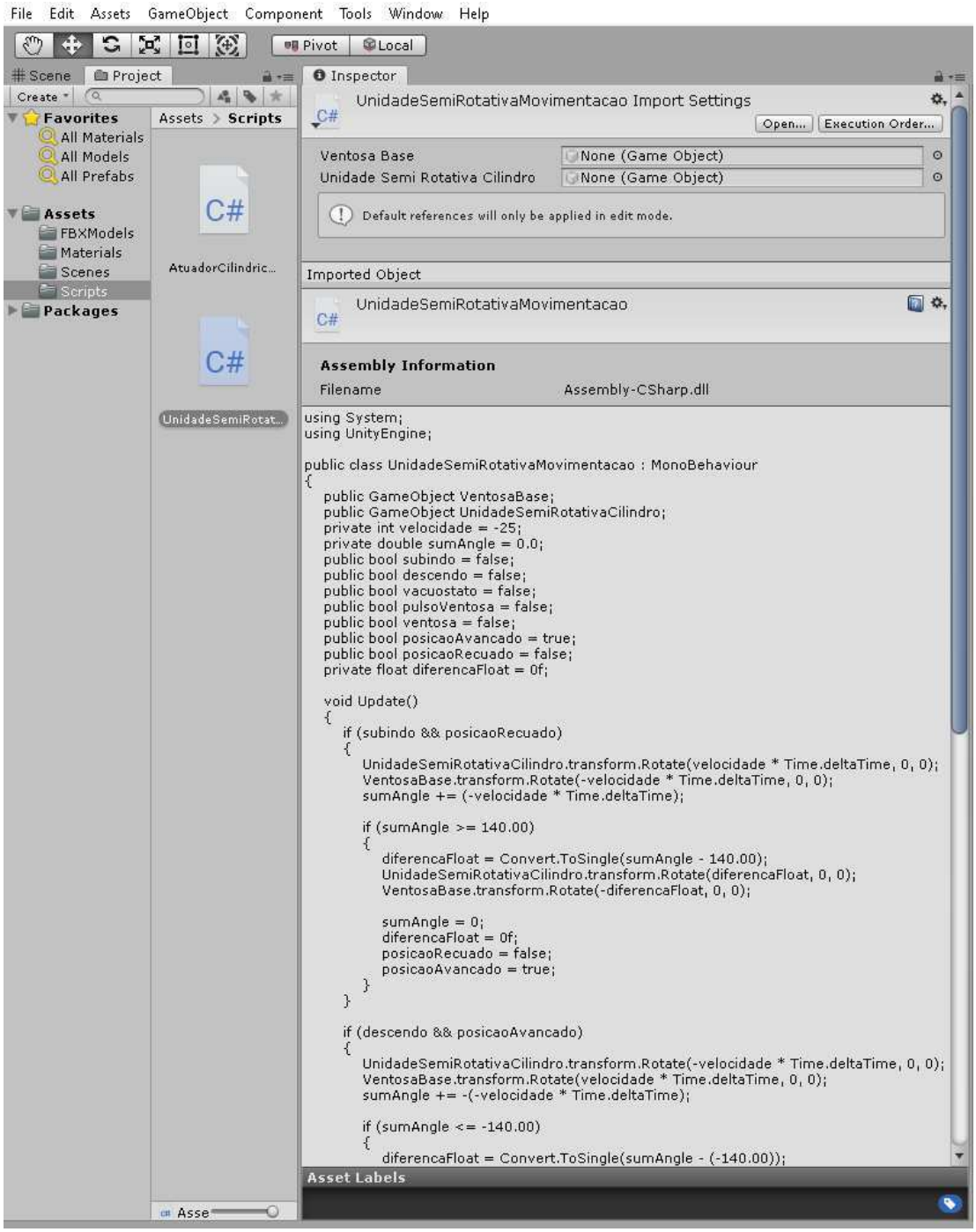
Fonte: Autoria própria.

Os mesmos passos foram realizados para o módulo de acionamento rotativo: cilindro de conexão da unidade semi-rotativa e ventosa. Neste caso, não tratou-se de um movimento de translação, mas de rotação.

De forma similar ao movimento de translação, o Unity3D disponibiliza diferentes métodos para sua realização. Dentre os métodos disponibilizados, utilizou-se o método Transform.Rotate, o qual rotaciona o objeto em torno do eixo definido dos graus definidos no código.

$$UnidadeSemiRotativaCilindro.Transform.Rotate(velocidade * Time.deltaTime, 0, 0);$$

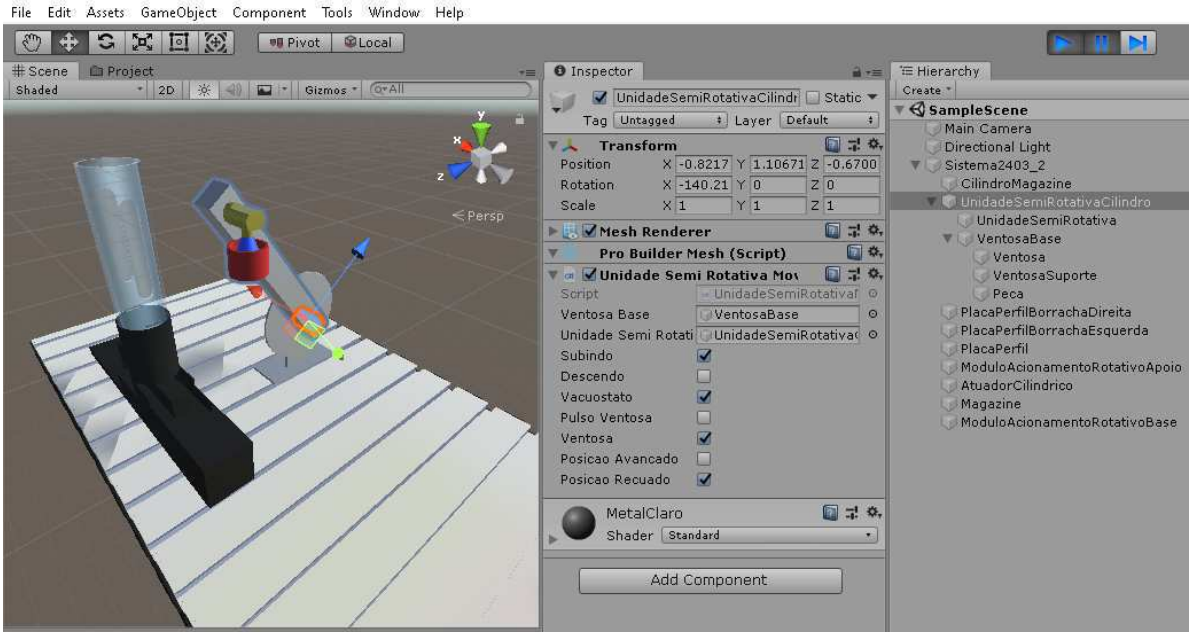
Figura 37 – Código inicial da dinâmica do módulo de acionamento rotativo.



Fonte: Autoria própria.

A integração dos dois códigos de implementação da dinâmica de movimentação para a estação de distribuição pode ser vista na figura 38. Além da simples translação e rotação dos componentes, utilizou-se também de um comando para inserir a peça na hierarquia de objetos conectados à ventosa, em uma camada inferior, de tal forma que a rotação desta implica em uma rotação de mesmo ângulo da peça, replicando o efeito de sucção à vácuo do sistema real. Ao final da rotação, a peça é liberada da ventosa, ao removê-la da hierarquia desta, o que permite que a peça caia, por ação gravitacional.

Figura 38 – Replicação da dinâmica da estação de distribuição.



Fonte: Autoria própria.

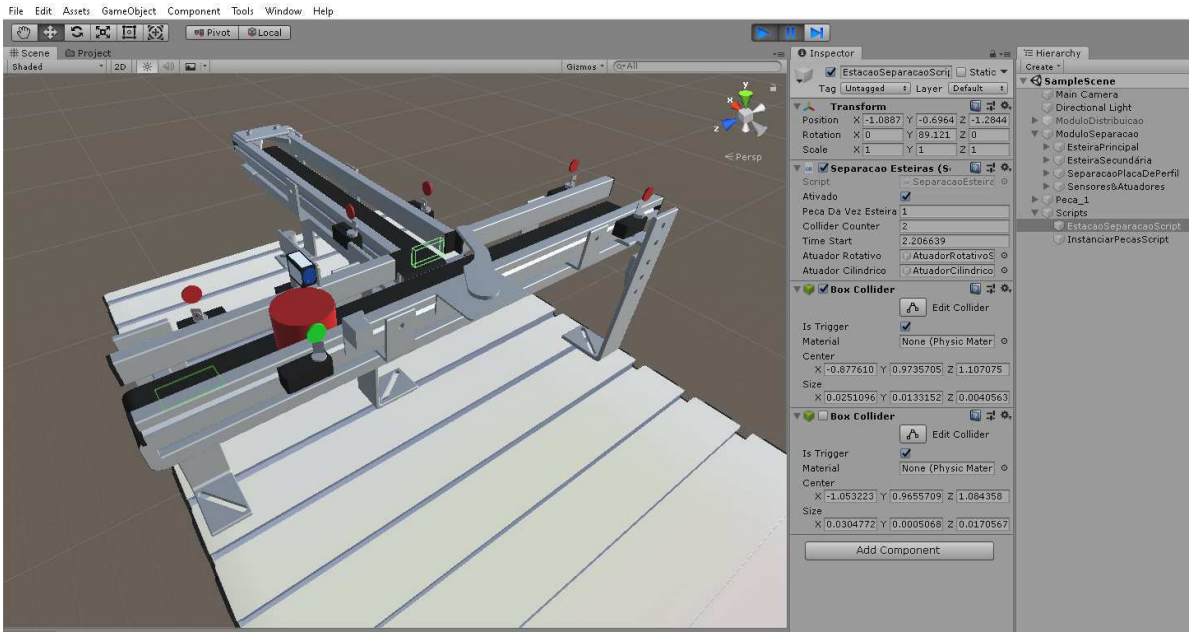
3.2.2 Estação de Separação

A dinâmica da estação de separação constitui-se, basicamente, da movimentação da peça ao longo da esteira principal, do acionamento do atuador cilíndrico que bloqueia a peça no ponto de verificação de sua orientação e do acionamento ou não do módulo de desvio, que desvia a peça para a esteira secundária, a depender de sua orientação. Para o acionamento destes componentes, faz-se necessário configurar também a lógica para os sensores presentes na estação.

Considerando a maior quantidade de interações entre os componentes da estação, optou-se por utilizar de colisores na esteira, atuadores, bloqueio final da esteira secundária e nos sensores, a fim de possibilitar uma dinâmica mais realista, e uma variável para controlar o tempo gasto pela peça em cada trecho. Para os sensores, os colisores atuaram como gatilhos de ativação da lógica de troca de estado destes. A dinâmica implementada para a estação é apresentada na figuras 39 e 40.

Ao cair por ação gravitacional, a peça ativa o colisor presente no início da esteira que define sua velocidade como constante, no eixo da esteira principal.

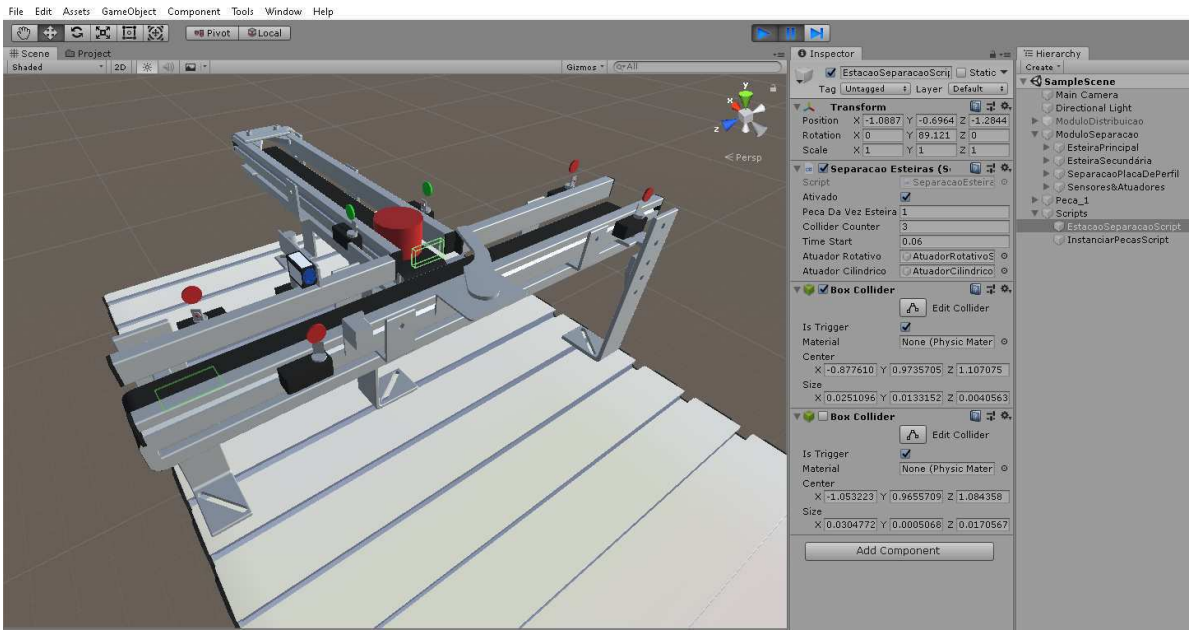
Figura 39 – Replicação da dinâmica da estação de separação - Peça bloqueada.



Fonte: Autoria própria.

Caso a peça tenha o furo virado para baixo, o módulo de desvio é acionado, desviando a peça para o segundo colisor que, ativado, atribuirá à peça uma velocidade constante no eixo da esteira secundária. Caso contrário, a peça segue na esteira principal.

Figura 40 – Replicação da dinâmica da estação de separação - Peça desviada.



Fonte: Autoria própria.

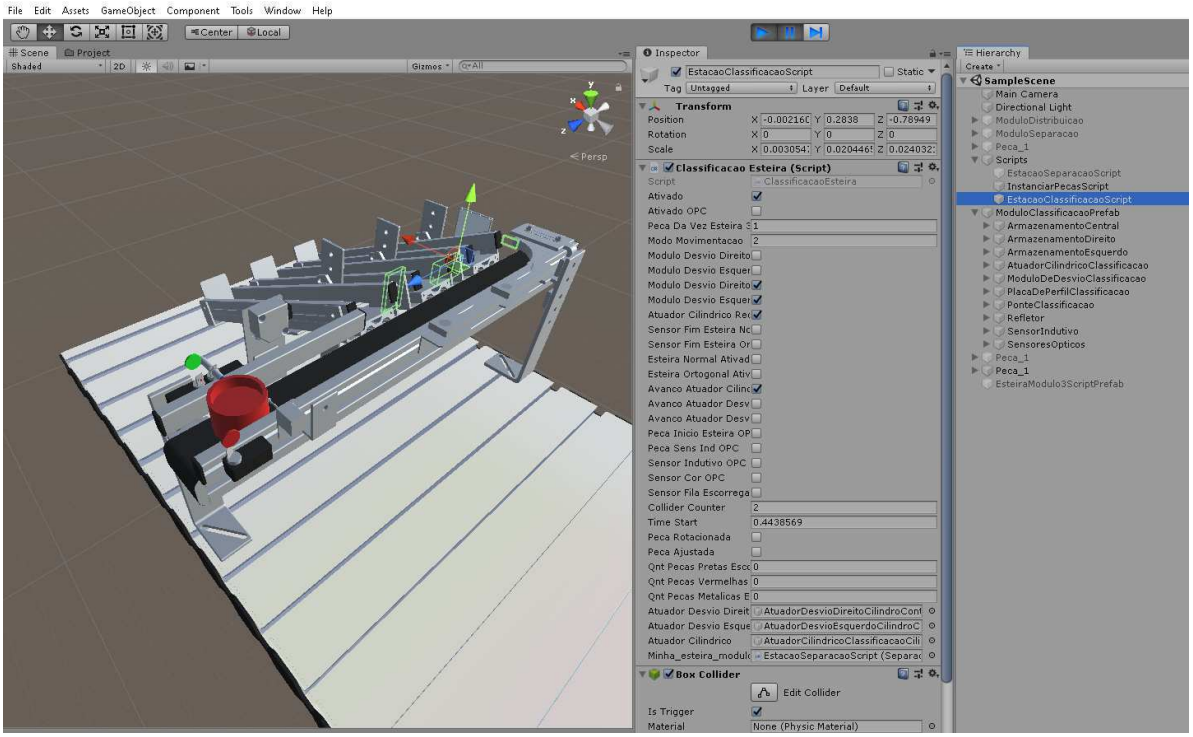
3.2.3 Estação de Classificação

A dinâmica da estação de classificação constitui-se da movimentação da peça ao longo de uma esteira única, do acionamento do atuador cilíndrico que bloqueia a peça no ponto de verificação de sua cor e material, como visto na figura 41, e do acionamento do respectivo módulo de desvio, a depender de sua cor e material. Após o desvio, a peça desliza em seu escorregador de armazenagem específico até parar em um anteparo final.

Para o acionamento destes componentes, utilizou-se de colisores e de uma variável para controlar o tempo gasto pela peça em cada trecho, de maneira similar à adotada para a estação de separação. Especialmente para a estação de classificação foi necessária uma maior atenção com relação aos intervalos de tempo gastos pela peça em cada trecho, pois, caso má configurados, levam a peça a ficar bloqueada na esteira antes de chegar a seu destino final.

Configurou-se a dinâmica da peça de tal forma que esta mantivesse uma velocidade constante no eixo da esteira, parasse ao colidir com o atuador cilíndrico avançado e retomasse a movimentação anterior, a partir do momento em que o atuador cilíndrico é recuado, como visto na figura 41.

Figura 41 – Detecção do material da peça no gêmeo digital.

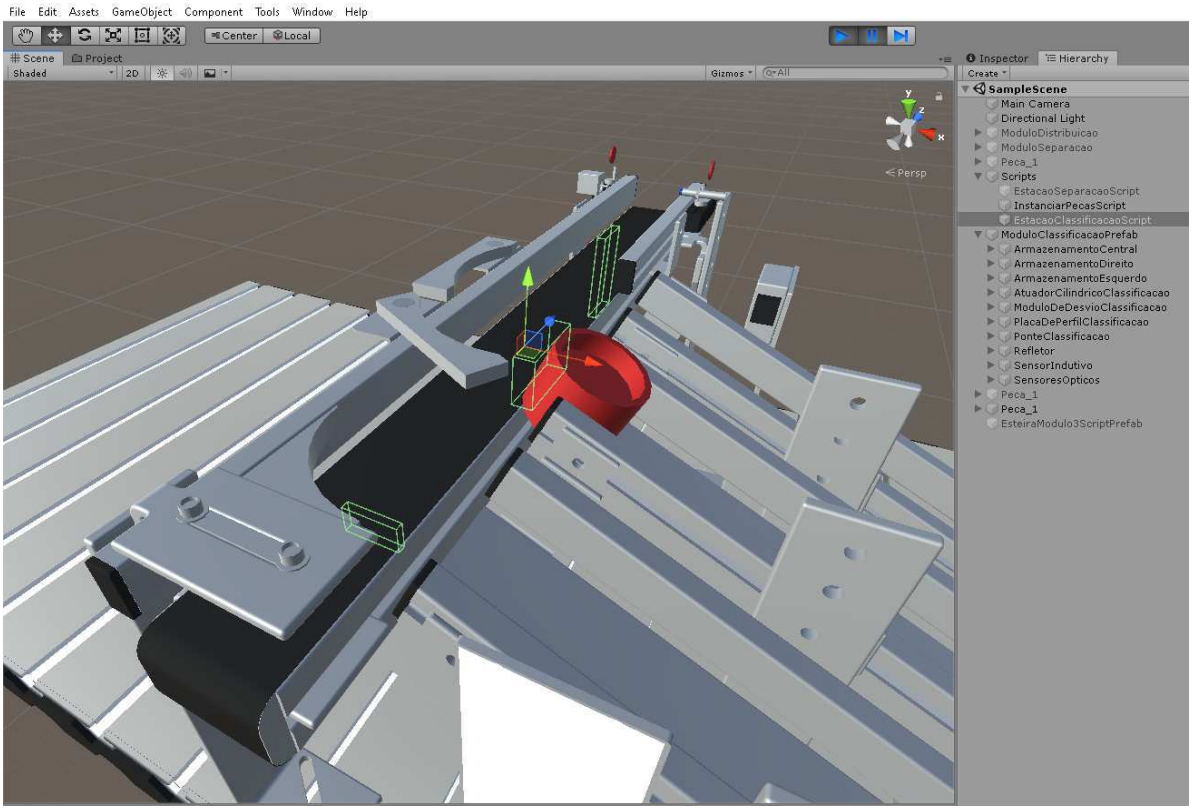


Fonte: Autoria própria.

A verificação da cor e do material da peça foi implementada checando a propriedade *material* do componente *Renderer* do objeto referente à peça. Por meio desta propriedade é possível obter qual material foi atribuído à peça. Após a verificação, é atribuída uma tag à peça, relativa ao resultado anterior, a qual é utilizada para determinar qual módulo de desvio será acionado.

De forma similar à estação de separação, o módulo de desvio respectivo é responsável por desviar a peça e fazê-la acionar um dos colisores inseridos no início de cada escorregador. Estes, por sua vez, são responsáveis por rotacionar a peça do ângulo do escorregador e configurar sua velocidade em uma componente vertical e uma horizontal, possibilitando uma dinâmica mais realista do deslizamento da peça sobre o escorregador, como é possível observar na figura 42.

Figura 42 – Classificação da peça no gêmeo digital.

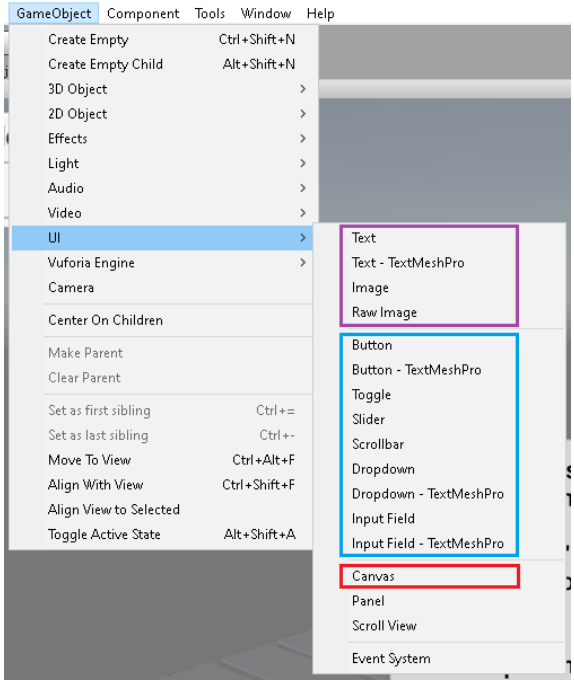


Fonte: Autoria própria.

3.3 Interface com o Usuário

Usando o menu **GameObject** → **UI**, é possível acessar a lista de elementos pré-configurados disponíveis de interface com o usuário que fazem parte do kit de ferramentas Unity UI. Estes elementos estão divididos nas categorias componentes visuais, de interação e canvas, apresentados em destaque na Figura 43 em roxo, azul e vermelho, respectivamente.

Figura 43 – Componentes visuais, de interação e canvas do Unity UI.



Fonte: Autoria própria.

O Unity UI é um kit de ferramentas de UI para o desenvolvimento de interfaces de usuário para jogos e aplicativos. O Unity UI é um sistema de UI baseado em *GameObject* que usa componentes e a visão de jogo para organizar, posicionar e aplicar estilo às interfaces de usuário (UNITY, a).

Os componentes visuais são responsáveis pela implementação da interface gráfica com o usuário e referem-se, basicamente aos componentes *Text* e *Image*. O componente *Text*, possui uma área para que se insira o texto a ser exibido. É possível definir a fonte, seu estilo e tamanho, além de configurações relativas ao alinhamento e redimensionamento do texto. O componente *Image*, por sua vez, é composto por um componente *Image* específico e um *Rect Transform* que o comporta, permitindo a inserção de imagens a partir da importação destas como sprites.

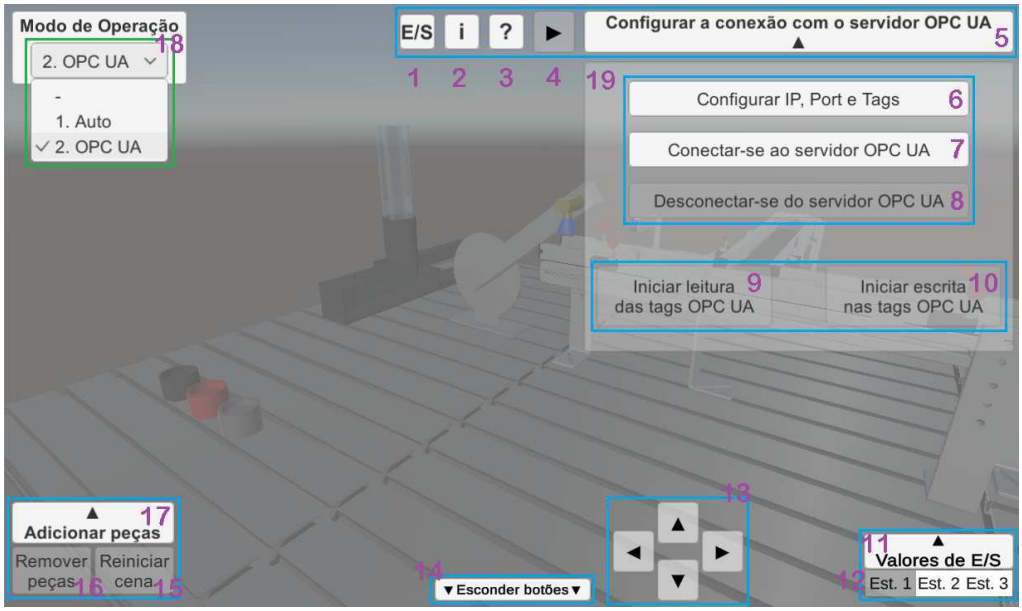
Os componentes de interação são responsáveis por tratar eventos de mouse ou toque e interação usando um teclado ou controlador. Estes componentes não são visíveis por si próprios e devem ser combinados com um ou mais componentes visuais para funcionar

corretamente. Além disso, os componentes de interação são selecionáveis, o que significa que compartilham a funcionalidade integrada para visualizar as transições entre os estados (normal, destacado, pressionado, desabilitado) e para navegar para outros selecionáveis usando o teclado ou controlador.

O Canvas, no Unity, é a área em que todos os elementos da UI devem inseridos. O Canvas é um *GameObject* com um componente Canvas nele, e todos os elementos de UI devem ser filhos desse Canvas, na hierarquia de objetos. Ao inserir-se um novo elemento de UI, como uma imagem usando o menu **GameObject** → **UI** → **Image**, por exemplo, o Unity cria automaticamente um Canvas, se ainda não houver um na cena e define o elemento *Image* como filho desse Canvas.

Os elementos da interface com o usuário utilizados na tela inicial da aplicação podem ser vistos na Figura 44, onde destaca-se os botões, em azul, e o menu suspenso, em verde. Ao selecionar o modo de operação "2. OPC UA", os elementos de 4 a 10 e 19 são exibidos para que o usuário possa realizar as configurações respectivas.

Figura 44 – Componentes de interface com o usuário da tela inicial.



Fonte: Autoria própria.

Os elementos de 1 a 3 referem-se a botões de auxílio informativo ao usuário.

1. Botão de visualização do painel dos mapas de E/S referentes à cada estação;
2. Botão de visualização do painel de boas-vindas aos usuários;
3. Botão de visualização do tutorial composto de 18 passos explicativos;
4. Botão de redução/expansão do botão 5;

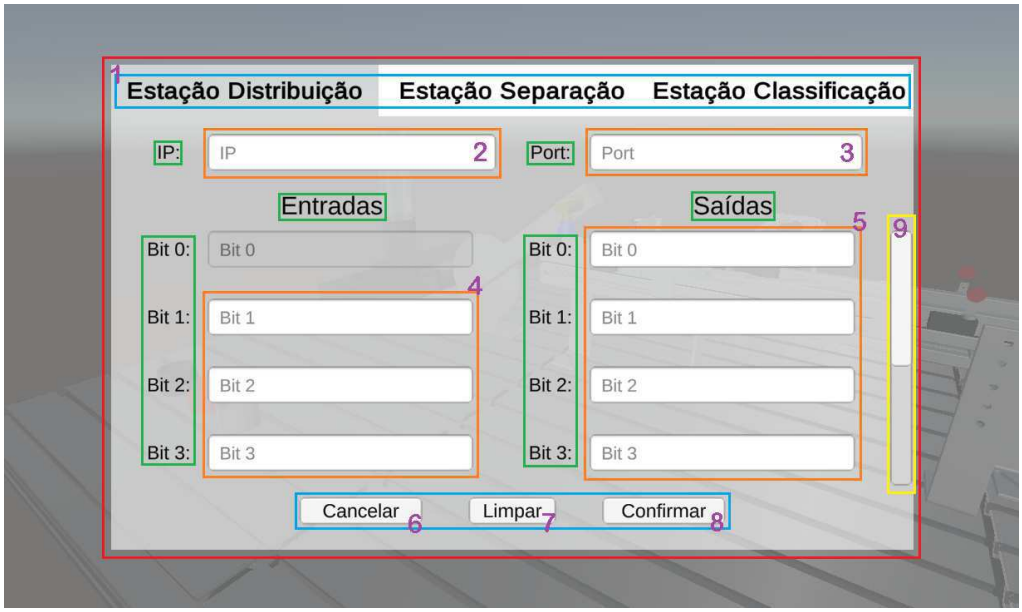
5. Botão de exibição/ocultamento do painel 19;

Os elementos de 6 a 10 referem-se aos botões de controle e configuração da comunicação entre a aplicação e o servidor OPC UA.

6. Botão de exibição do painel de configuração da conexão com o servidor OPC UA (IP, *Port* e *tags*);
7. Botão de estabelecimento da conexão com o servidor OPC UA configurado;
8. Botão de desconexão do servidor OPC UA;
9. Botão de estabelecimento da função de leitura das tags OPC UA configuradas;
10. Botão de estabelecimento da função de escrita nas tags OPC UA configuradas;
11. Botão de exibição/ocultamento do painel de visualização dos estados das variáveis de E/S;
12. Botões de seleção da estação referida no painel exibido a partir do botão 11, onde Est. 1, 2 e 3 referem-se, respectivamente às estações de distribuição, separação e classificação;
13. Botões de movimentação da câmera;
14. Botão de exibição/ocultamento dos demais botões presente na interface;
15. Botão de reinício da cena da aplicação, o qual restaura o valor inicial de todas as variáveis;
16. Botão de remoção das peças, o qual permite deletar as peças inseridas no cilindro do magazine antes de iniciar a execução do sistema;
17. Botão de exibição/ocultamento do painel de inserção das peças no cilindro do magazine;
18. Menu suspenso de seleção do modo de operação do sistema, onde a opção 1 refere-se ao modo automático e a opção 2 ao modo conectado com o servidor OPC UA;
19. Painel que comporta os elementos de 6 a 10.

Os elementos da interface com o usuário utilizados para a definição das *tags* OPC UA criadas e configuração da conexão da aplicação com o servidor OPC UA podem ser vistos na figura 45, onde destacam-se os botões, em azul, os campos de inserção de texto, em laranja, os textos, em verde, a barra de rolagem, em amarelo, e o painel, em vermelho. Ao clicar no botão 6, referido anteriormente, uma animação de *pop-up* do painel é executada, assim como são ocultados os elementos da interface com o usuário presentes na tela inicial, permitindo um maior destaque do painel.

Figura 45 – Componentes de interface com o usuário do painel de configuração OPC UA.



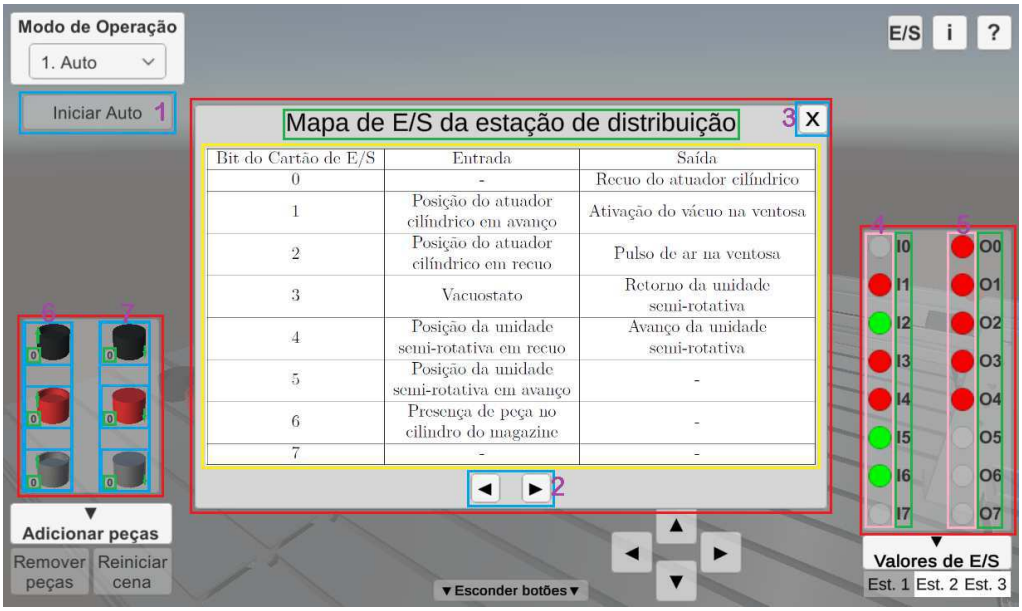
Fonte: Autoria própria.

1. Botões de seleção de configuração dos bits de E/S referentes a cada estação;
2. Campo de inserção de texto referente ao IP do servidor OPC UA;
3. Campo de inserção de texto referente à porta do servidor OPC UA disponibilizada;
4. Campos de inserção de texto referentes aos endereços das tags OPC UA correspondentes aos bits de entrada;
5. Campos de inserção de texto referentes aos endereços das tags OPC UA correspondentes aos bits de saída;
6. Botão de cancelamento da configuração de conexão com o servidor OPC UA, o qual fecha o painel sem salvar as alterações realizadas nos campos;
7. Botão de limpeza dos campos de inserção de texto da estação atual selecionada;
8. Botão de confirmação das alterações nos campos de inserção de texto, o qual salva as alterações para todas as três estações e fecha o painel;
9. Barra de rolagem, a qual permite ao usuário acessar os demais campos de inserção de texto, referentes aos bits de 4 a 7.

Os elementos da interface com o usuário utilizados nos painéis de visualização dos mapas de E/S, e dos valores assumidos por estas variáveis na aplicação, assim como o painel de inserção de peças são apresentados na Figura 46, onde destacam-se os botões,

em azul, a imagem, em amarelo, os textos, em verde, os *toggles*, em rosa, assim como os painéis, em vermelho.

Figura 46 – Componentes de interface com o usuário dos painéis auxiliares.



Fonte: Autoria própria.

1. Botão de início da execução da aplicação no modo automático;
2. Botões de seleção do mapa de E/S referente a cada estação;
3. Botão de fechamento do painel dos mapas de E/S;
4. *toggles* referentes aos valores das variáveis da aplicação correspondentes às entradas do sistema, para a estação selecionada;
5. *toggles* referentes aos valores das variáveis da aplicação correspondentes às saídas do sistema, para a estação selecionada;
6. Botões de inserção das peças com orientação do furo para cima no cilindro do magazine;
7. Botões de inserção das peças com orientação do furo para baixo no cilindro do magazine.

Os *toggles* em 4 e 5 foram desativados de forma a funcionarem apenas como indicadores de valor. Além disso, pôde-se aproveitar de sua funcionalidade baseada em eventos, a fim de otimizar a aplicação no modo de operação conectado ao servidor OPC UA, o que será detalhado na seção 3.5.

A região de clique dos botões em 6 e 7 foi substituída por imagens referentes às respectivas peças inseridas, de forma a auxiliar o usuário no entendimento de sua funcionalidade. Inseriu-se também setas ao lado dos botões, orientadas para cima ou para baixo (Figuras 47a e 47b), de maneira a auxiliar o usuário na identificação da orientação respectiva da peça selecionada.

Figura 47 – Botões de inserção das peças modificados.

(a) Botão peça vermelha com furo para cima. (b) Botão peça metálica com furo para baixo.

Fonte: Autoria própria.

3.4 Integração com recursos de Realidade Aumentada

O Unity 3D permite a utilização de diversos pacotes de *plugins* de realidade aumentada em seu ambiente, entretanto o Vuforia Engine é a plataforma mais amplamente usada para o desenvolvimento de AR, oferecendo suporte para os principais celulares, tablets e óculos. Com este, é possível adicionar facilmente funcionalidades avançadas de visão computacional aos aplicativos Android, iOS e UWP, para criar experiências de RA que interagem de forma realista com objetos e o ambiente (UNITY, b).

O Vuforia usa tecnologia de visão computacional para reconhecer e rastrear imagens planas e objetos 3D em tempo real. Esse recurso de registro de imagem permite que os desenvolvedores posicionem e orientem objetos virtuais, como modelos 3D, em relação a objetos do mundo real quando são visualizados pela câmera de um dispositivo móvel. O objeto virtual, então, rastreia a posição e orientação da imagem em tempo real para que a perspectiva do visualizador no objeto corresponda à perspectiva no alvo. Isto permite com que o objeto virtual pareça ser uma parte da cena do mundo real (PTC, 2020).

Figura 48 – Logo do Vuforia.

Fonte: PTC

O SDK Vuforia suporta uma variedade de tipos de alvo 2D e 3D, incluindo alvos de imagem "sem marcador". Recursos adicionais do SDK incluem 6 graus de liberdade de localização do dispositivo no espaço, detecção de oclusão localizada usando "botões

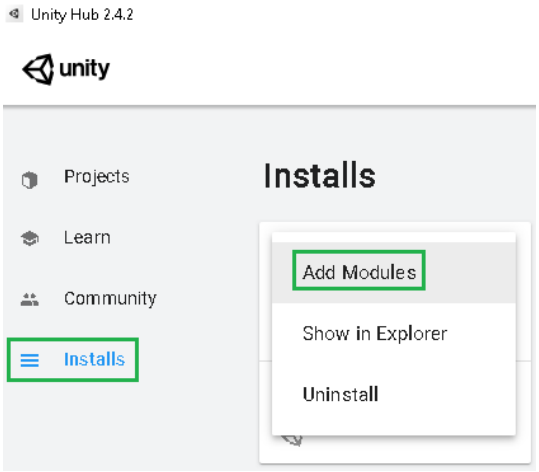
virtuais", seleção de alvo de imagem em tempo de execução e a capacidade de criar e reconfigurar conjuntos de alvos em tempo de execução de forma programada.

O Vuforia permite a utilização de suas funcionalidades de forma gratuita, desde que se tenha uma conta cadastrada em sua plataforma, além da notificação de que sua logo sempre aparecerá como marca d'água na visualização do aplicativo.

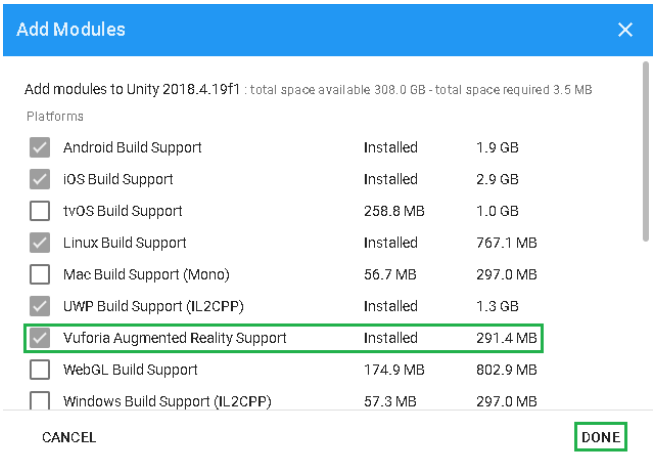
O Vuforia para o Unity versão 2018.4.19f1 (LTS) pode ser instalado no ambiente como um módulo. Para isto, basta acessar o **Unity Hub** → **Installs** e, na versão instalada do Unity, acessar ⋮ → **Add modules** (Figura 49a). Após isto, basta marcar a opção Vuforia Augmented Reality Support e clicar em *Done* para instalar o módulo (Figura 49b).

Figura 49 – Instalação do módulo Vuforia no Unity 2018.4.19f1.

(a) Acesso aos módulos do Unity.



(b) Instalação do módulo Vuforia.

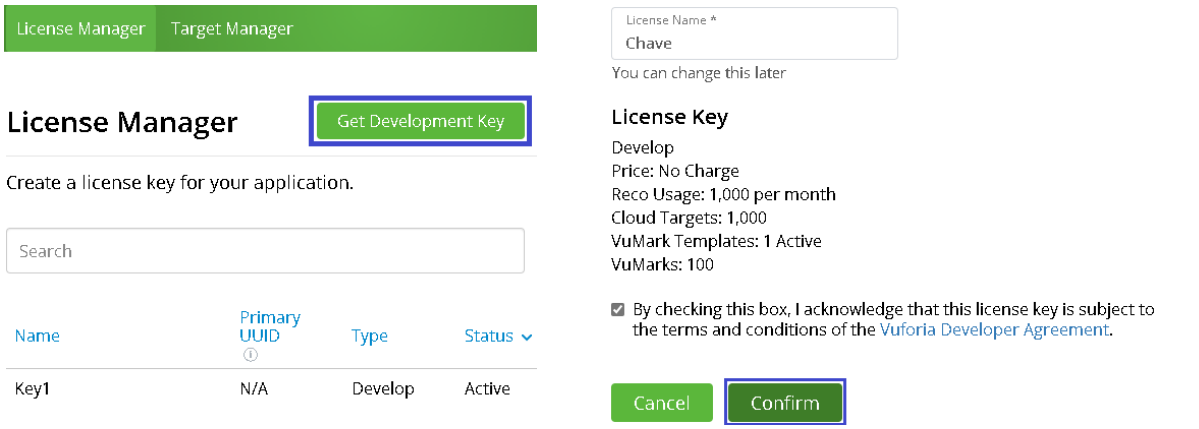


Fonte: Autoria própria.

Após a instalação do módulo no Unity e criação de uma conta na plataforma do Vuforia, é necessário obter uma chave de desenvolvedor para utilizar de suas funcionalidades, o que é feito no ambiente de desenvolvedores do site do Vuforia, na aba *License Manager*. Ao clicar na opção *Get Development Key*, é possível definir um nome para a chave e criá-la, clicando em *Confirm*, como visto nas figuras 50a e 50b abaixo.

Figura 50 – Obtenção de uma chave de desenvolvimento do Vuforia.

(a) Gerenciador de licenças. (b) Obtenção de uma nova chave de licença.

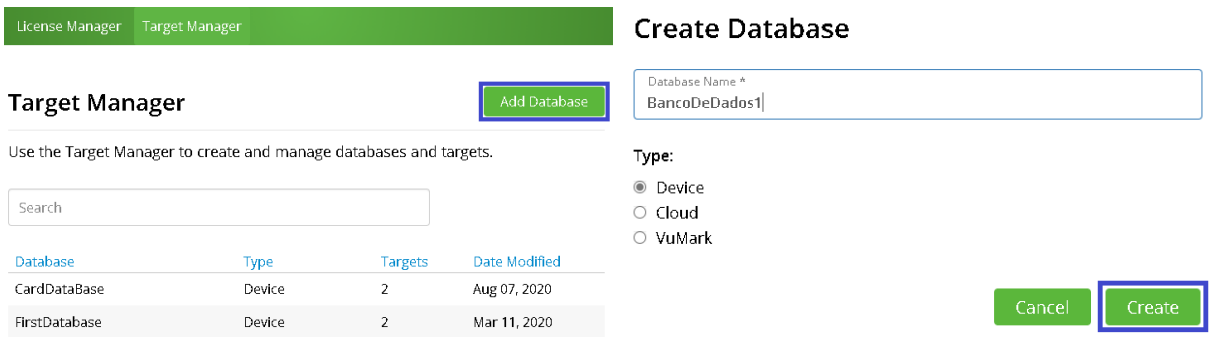


Fonte: Adaptado de PTC.

Tendo obtido uma chave de desenvolvimento, resta definir os alvos a serem utilizados na aplicação. Inicialmente, é necessário criar um banco de dados para armazenar os alvos. Para isso, basta acessar a aba *Target Manager*, no mesmo ambiente, e clicar em *Add Database* 51a. Deve-se, então, definir o nome do banco de dados e o seu tipo 51b, escolhendo o tipo *Device*, o qual fornece ao aplicativo em RA um banco de dados de alvos acessível localmente.

Figura 51 – Configuração dos alvos de RA utilizados na aplicação.

(a) Acesso ao *Target Manager*. (b) Adição de banco de dados de alvos.

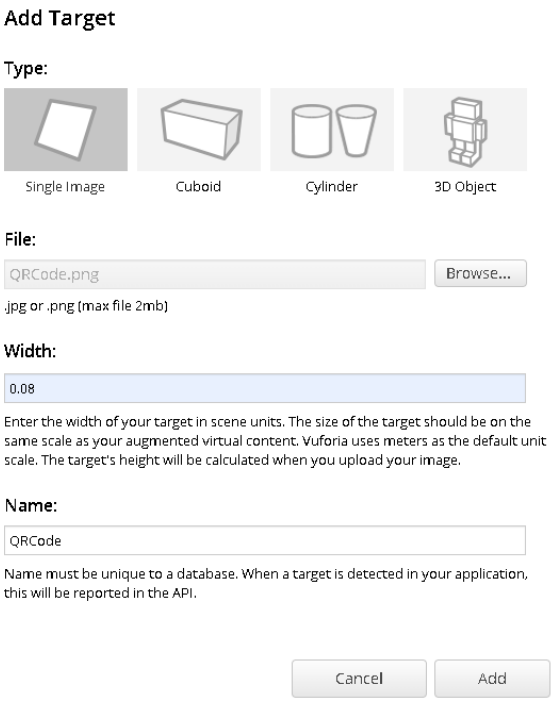


Fonte: Adaptado de PTC.

Ao clicar no nome do banco de dados criado, é possível inserir novos alvos a serem utilizados na aplicação. Para isto, basta clicar em *Add Target* e em *Single Image* para adicionar um alvo de imagem. Para este tipo de alvo, a fim de melhorar a estabilidade do conteúdo virtual no aplicativo e o rastreamento, é importante informar aproximadamente largura da imagem real utilizada como alvo, em metros, pois 1 unidade de espaço no Unity equivale a 1 metro. Desta forma, dado que a imagem real deverá ter, aproximadamente 8 centímetros, inseriu-se 0.08 no campo de largura.

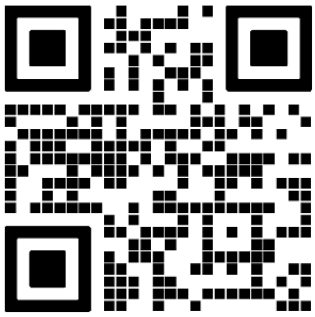
Figura 52 – Inserção de alvos de RA no banco de dados.

(a) Configurações do alvo.



Fonte: Adaptado de PTC.

(b) Código QR utilizado.

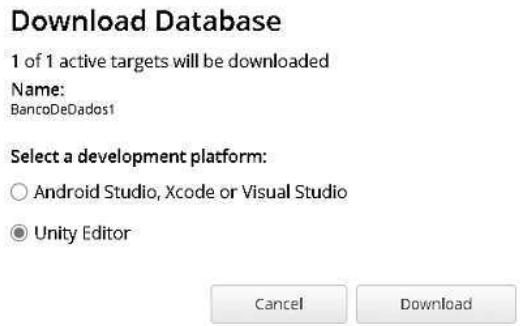


Fonte: Adaptado de QR Code Generator.

Após inserir o alvo no banco de dados, a fim de adicioná-lo ao ambiente do Unity, é preciso, inicialmente realizar o *download* deste. Para isto, basta clicar em *Download Database* (*All*) e definir a plataforma de desenvolvimento como Unity Editor 53a. Feito isto, basta importar o pacote no Unity Editor acessando a aba **Assets**, e então escolher **Import** → **Custom Package** 53b. O pacote importará automaticamente os arquivos do banco de dados para a pasta StreamingAssets/Vuforia do projeto.

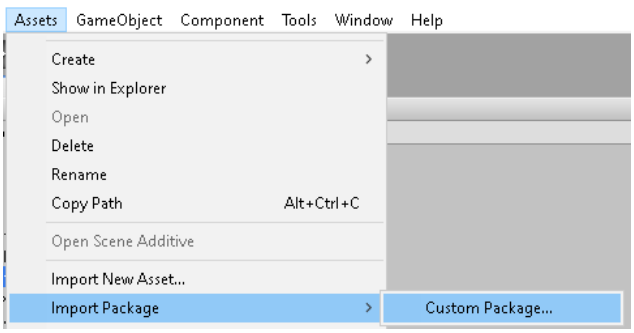
Figura 53 – Importação do banco de dados de alvos no ambiente do Unity.

(a) *Download* do banco de dados.



Fonte: Adaptado de PTC.

(b) Importação do pacote no Unity.

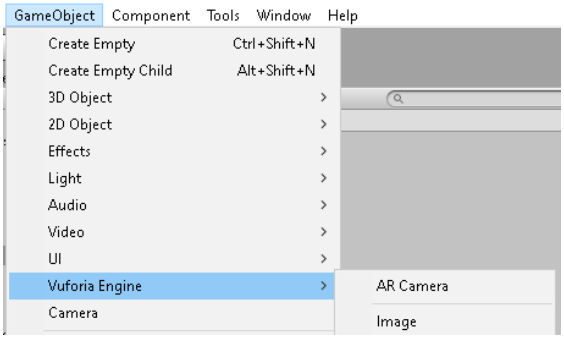


Fonte: Autoria própria.

Para utilizar, no ambiente do Unity, o alvo de imagem configurado anteriormente, é necessário adicionar ao projeto os *gameobjects* AR *Camera* e *Image* do Vuforia. Para isto,

basta acessar a aba **GameObject**, no Unity Editor, e então escolher **Vuforia Engine** → **Image** e **AR Camera** (Figura 54).

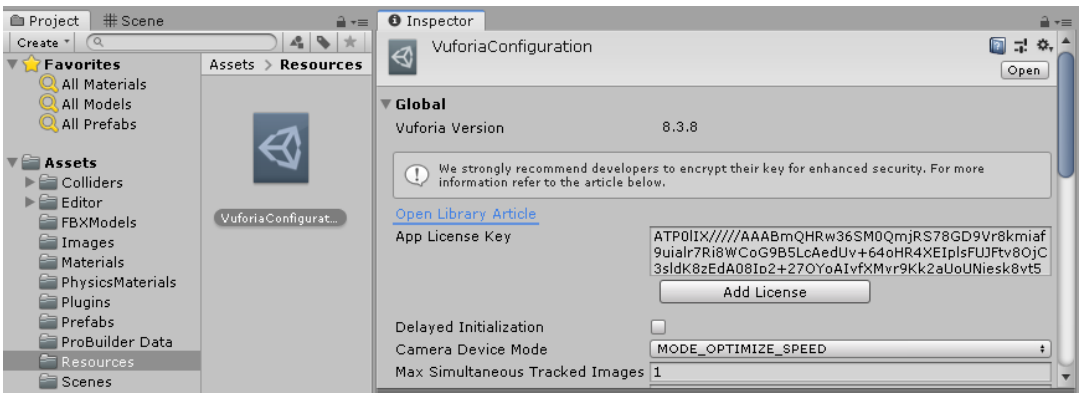
Figura 54 – Inserção dos *gameobjects* AR *Camera* e *Image* do Vuforia Engine.



Fonte: Autoria própria.

A fim de habilitar a utilização dos recursos do Vuforia no Unity é necessário informar o valor da chave de licença obtida anteriormente, dentro do ambiente. Para isto, basta acessar a pasta Assets/Resources e, em VuforiaConfiguration, inserir o valor da chave de licença.

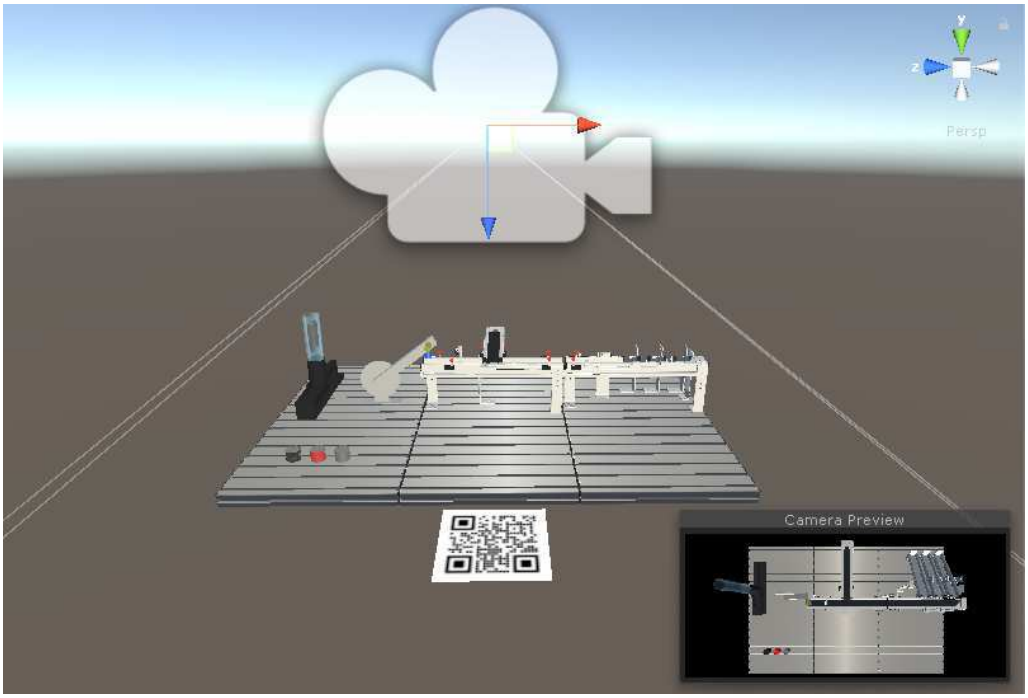
Figura 55 – Inserção do valor da chave de licença do Vuforia no Unity.



Fonte: Autoria própria.

Após isto, basta posicionar o *gameobject Image*, com o código QR, logo abaixo do sistema e o *gameobject* AR *Camera* acima do sistema, com seu campo de visão voltado para baixo e focado no sistema, de forma a comportar todos os seus componentes.

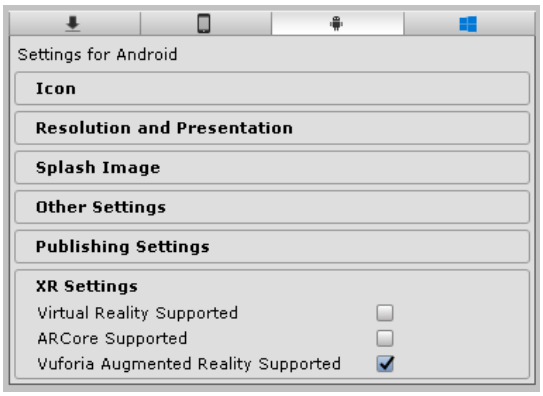
Figura 56 – Ajuste da câmera AR e do alvo de imagem no Unity.



Fonte: Autoria própria.

Finalmente, para realizar o *build* do aplicativo com os recursos integrados de realidade aumentada do Vuforia basta habilitá-los em **Edit** → **Project Settings** e, na plataforma alvo, selecionar a opção **Vuforia Augmented Reality Supported**

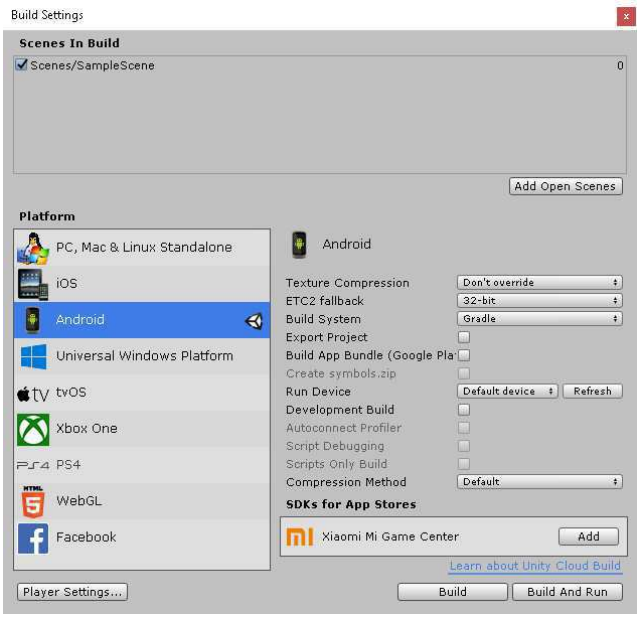
Figura 57 – Habilitação dos recursos do Vuforia para a aplicação.



Fonte: Autoria própria.

Finalizadas as configurações anteriores, para realizar o *build* do aplicativo na plataforma alvo basta acessar **File** → **Build Settings** e escolher a opção *Build*.

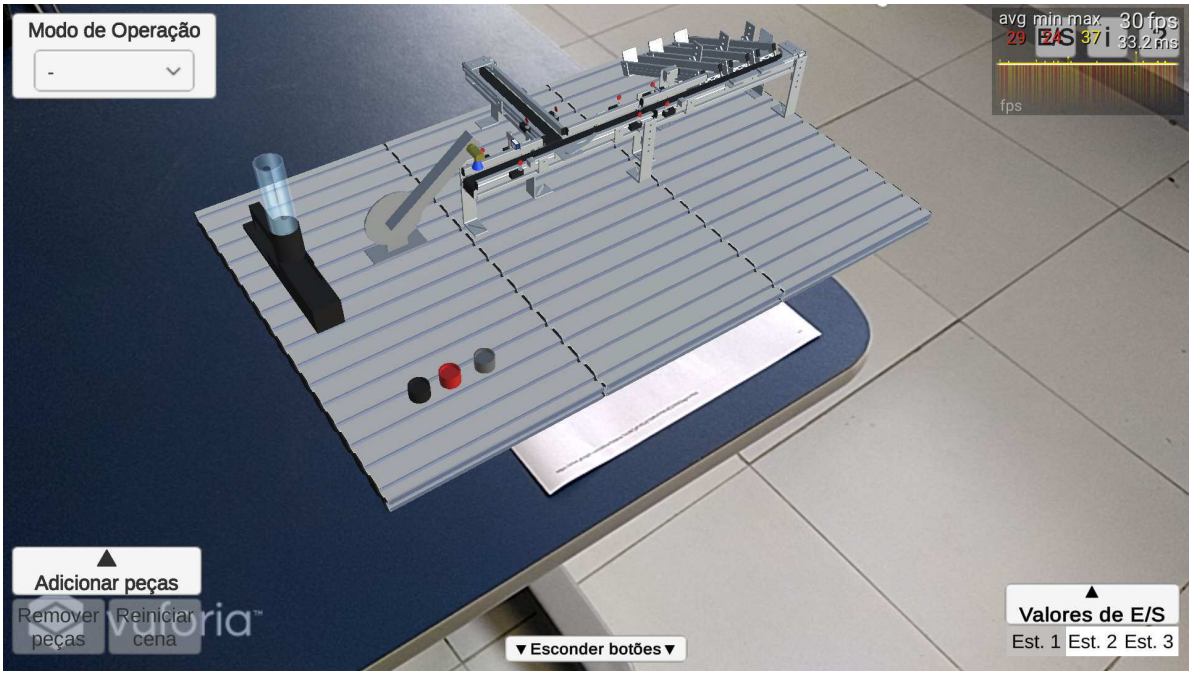
Figura 58 – *Build* do aplicativo para o sistema operacional Android.



Fonte: Autoria própria.

Ao apontar a câmera do *smartphone* para o alvo de imagem, todos os *gameobjects* inseridos dentro do campo de visão da *AR camera* serão exibidos, tendo como referência o código QR utilizado.

Figura 59 – Componentes 3D do sistema exibidos em realidade aumentada.



Fonte: Autoria própria.

3.5 Comunicação com o servidor OPC UA

O Unity tem como objetivo oferecer suporte à grande maioria das APIs no perfil do .NET Standard 2.0 em todas as plataformas. Embora nem todas as plataformas sejam totalmente compatíveis com o .NET Standard, as bibliotecas que visam a compatibilidade entre plataformas devem ter como alvo o perfil do .NET Standard 2.0 (UNITY, 2019).

A partir do .NET Standard 2.0, o modo de compatibilidade do .NET Framework foi introduzido, o qual permite que os projetos .NET Standard referenciem bibliotecas .NET Framework como se fossem compilados para .NET Standard (MICROSOFT, 2020). Nesse sentido, embora haja bibliotecas compiladas para .NET Standard e .NET Framework, a aplicação é capaz de prover, para cada plataforma, um netstandard.dll que direciona as definições para a dll correta no framework.

Plugins de código gerenciado compilados fora do Unity podem funcionar com o perfil .NET Standard 2.0 ou .NET 4.x no Unity. A Tabela 2 descreve as configurações que o Unity suporta:

Tabela 2 – Suporte ao .NET no Unity.

	Nível de compatibilidade da API	
	.NET Standard 2.0	**.NET 4.x**
Plugin compilado em:		
.NET Standard	Suportado	Suportado
.NET Framework	Limitado	Suportado

Fonte: Adaptado de Unity.

É possível identificar que *plugins* gerenciados compilados em qualquer versão do .NET Standard funcionam com o Unity. O suporte limitado indica que o Unity oferece suporte à configuração se todas as APIs usadas do .NET Framework estiverem presentes no perfil do .NET Standard 2.0.

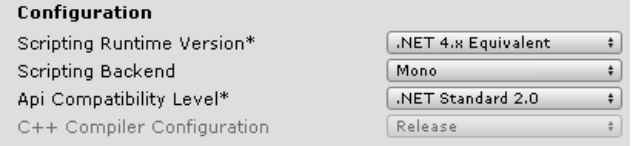
Embora o Unity ofereça suporte a outros perfis de API do .NET, recomenda-se utilizar o nível de compatibilidade da API do .NET Standard 2.0 também pelos seguintes motivos (UNITY, 2020):

- O .NET Standard 2.0 tem melhor suporte multiplataforma, portanto, é mais provável que o código funcione em todas as plataformas.
- O .NET Standard 2.0 é compatível com todos os ambientes de execução .NET, portanto, o código funciona em mais ambientes (.NET Framework. .NET Core, Xamarin, Unity).

- O .NET Standard move mais erros para o tempo de compilação. Várias APIs no .NET 4.7.1 estão disponíveis em tempo de compilação, mas há implementações em algumas plataformas que lançam uma exceção em tempo de execução.

Para definir o nível de compatibilidade da API no Unity, basta acessar, no ambiente do Unity Editor, **Edit** → **Project Settings** → **Player** → **Other Settings**, e selecionar o nível desejado no menu suspenso correspondente, como na Figura 60.

Figura 60 – Definição do nível de compatibilidade da API no Unity.



Fonte: Autoria própria.

Portanto, com o objetivo de implementar um cliente OPC UA, escrito em C#, na aplicação, de forma compatível para diversas plataformas, buscou-se por um SDK baseado em .NET Standard 2.0. Entre as opções disponíveis, o SDK da organização OPC Foundation foi selecionado por estar disponível de forma gratuita em seu repositório público.

Nesse repositório há diversos exemplos de clientes OPC UA construídos a partir do SDK. A partir desses exemplos, foi possível desenvolver uma versão do cliente mais simples e otimizada para a aplicação, permitindo a conexão com o servidor e a leitura e escrita de valores em nós.

Embora fosse possível desenvolver o cliente dentro do próprio projeto da aplicação, no Visual Studio, optou-se por desenvolvê-lo em um projeto separado, do tipo biblioteca de classes, para que fosse utilizado dentro da aplicação como uma *dynamic-link library* (DLL).

Inicialmente, criou-se um novo projeto no Visual Studio e, por meio do comando *Install-Package OPCFoundation.NetStandard.Opc.Ua* no Gerenciador de Pacotes, procedeu-se com a instalação dos pacotes do SDK. Após a finalização do projeto, pôde-se obter a dll relativa ao cliente desenvolvido.

Para o Unity, inicialmente criou-se a pasta "*Plugins*", dentro da pasta Assets, e nela inseriu-se a dll relativa ao cliente desenvolvido, assim como assim como as bibliotecas OPC UA utilizadas diretamente pelo cliente. Entretanto, dado que essas bibliotecas referenciam outras, requeridas para a execução do cliente, foi necessário também inserí-las na pasta Plugins. Isso foi feito a partir da pasta Release do projeto do cliente, onde se encontravam as bibliotecas e das mensagens de erro no Unity, que informaram os nomes das bibliotecas requeridas.

Finalizada a etapa de desenvolvimento do cliente, prosseguiu-se para a etapa de criação dos componentes de interface com o usuário relativos à comunicação OPC UA, presentes na Figura 61, assim como as classes e métodos correspondentes às funções implementadas.

Figura 61 – Botões da aplicação relativos à comunicação por meio do padrão OPC UA.

Fonte: Autoria própria.

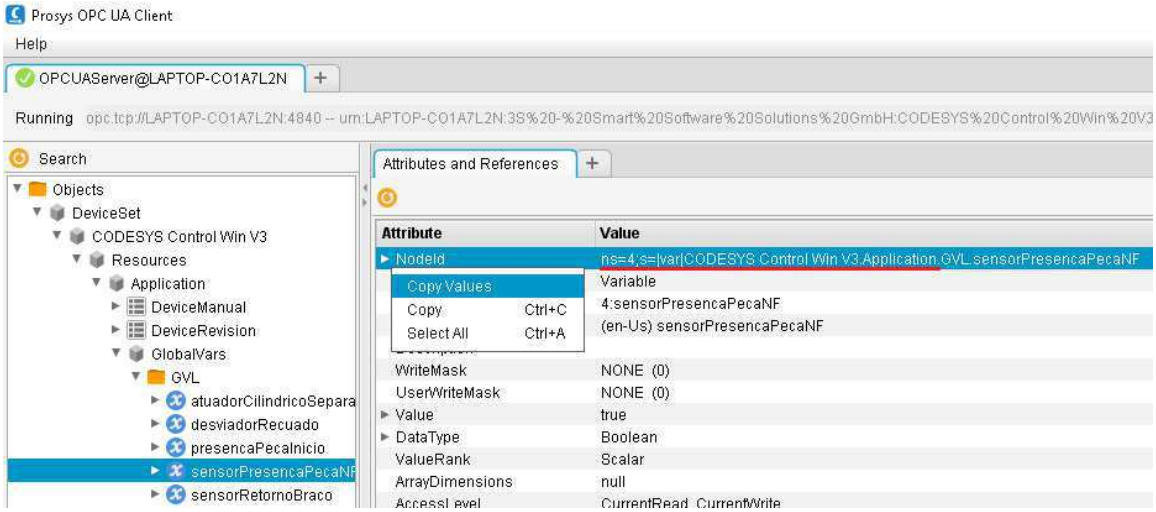
Inicialmente, implementou-se a funcionalidade do botão 1, o qual habilita o painel da Figura 62, relativo à configuração da conexão com o servidor OPC UA, que permite que o usuário insira os dados referentes ao IP, onde servidor está sendo executado, o número da porta disponibilizada, assim como os trechos especificados dos identificadores dos nós correspondentes aos bits de entrada e saída.

Figura 62 – Painel de configuração da conexão com o servidor OPC UA.

Fonte: Autoria própria.

A fim de obter de forma simples os identificadores dos nós, é possível utilizar um *software* como o Prosys OPC UA Client, para copiar os valores dos identificadores, como na Figura 63, e então inserí-los nos campos respectivos do painel de configuração na aplicação.

Figura 63 – Cópia dos identificadores dos nós no Prosys OPC UA Client.



Fonte: Autoria própria.

Ao utilizar o *software* CODESYS para implementar a execução do servidor na máquina, assim como o projeto baixado no CLP (virtual ou físico), os nós configurados possuem em comum o trecho inicial do seu identificador, destacado nas Figuras 63 e 64, o qual, com o objetivo de diminuir o tempo gasto no preenchimento do painel de configuração, deve ser removido na aplicação, pois já foi definido no código.

Figura 64 – Simplificação da inserção dos identificadores dos nós na aplicação.

```
#region CilindroMagazineVazio - Entrada 6
0 references
public void CilindroMagazineVazio()
{
    if(ButtonOPCDisconnectButton.interactable)
    {
        if (ToggleCilindroMagazineVazio.isOn)
        {
            // ------------------------------- Entrada 6 ------------------------------- //
            myOPCUAServer.EscreveNasTags("ns=4;s=|var|CODESYS Control Win V3.Application." +
                InputDistribuicaoEntradaBit6.text, false);
        }
        if (!ToggleCilindroMagazineVazio.isOn)
        {
            // ------------------------------- Entrada 6 ------------------------------- //
            myOPCUAServer.EscreveNasTags("ns=4;s=|var|CODESYS Control Win V3.Application." +
                InputDistribuicaoEntradaBit6.text, true);
        }
    }
}
#endregion
```

Fonte: Autoria própria.

Após isso, implementou-se a funcionalidade do botão 2, o qual refere-se à conexão do cliente, na aplicação, com o servidor. No momento em que esse botão é clicado, o método OPCConnect, na Figura 65 é executado. Inicialmente, as *strings* inseridas no painel de configuração referentes aos bits de saída são adicionadas a uma coleção de *tags* utilizando a classe *Dictionary*, como coleção do tipo chave/valor.

Figura 65 – Adição das *tags* configuradas em uma lista.

```
public void OPCConnect()
{
    try
    {
        TagList = new Dictionary<String, OPCUAClass.TagClass>();
        string rep = "|var|CODESYS Control Win V3.Application.";
        if (TagList == null)
        {
            try
            {
                TagList.Add("atuadorCilindrico",
                    new OPCUAClass.TagClass("atuadorCilindrico", rep + InputDistribuicaoSaidaBit0.text));
                TagList.Add("ventosa",
                    new OPCUAClass.TagClass("ventosa", rep + InputDistribuicaoSaidaBit1.text));
                TagList.Add("pulsoAr",
                    new OPCUAClass.TagClass("pulsoAr", rep + InputDistribuicaoSaidaBit2.text));
                TagList.Add("retornoBraco",
                    new OPCUAClass.TagClass("retornoBraco", rep + InputDistribuicaoSaidaBit3.text));
                TagList.Add("avancoBraco",
                    new OPCUAClass.TagClass("avancoBraco", rep + InputDistribuicaoSaidaBit4.text));
```

Fonte: Autoria própria.

Esse dicionário é composto de uma *string* como chave e da classe TagClass, implementada na classe do cliente OPC, como valor. O construtor dessa classe, na Figura 66, por sua vez, recebe como argumentos duas *strings*, sendo a primeira referente ao *displayName* e a segunda referente ao *nodeId* da tag adicionada.

Figura 66 – Construtor e métodos da classe TagClass.

```
public class TagClass
{
    public TagClass(string displayName, string nodeID)
    {
        DisplayName = displayName;
        NodeID = nodeID;
    }

    public DateTime LastUpdatedTime { get; set; }
    public DateTime LastSourceTimeStamp { get; set; }
    public string StatusCode { get; set; }

    public string LastGoodValue { get; set; }
    public string CurrentValue { get; set; }
    public string NodeID { get; set; }
    public string DisplayName { get; set; }
}
```

Fonte: Autoria própria.

Após a adição das *tags*, o mesmo método instancia a classe OPCUAClass, como visto na Figura 67, implementada no projeto do cliente OPC, por meio de seu construtor. Como argumentos são passadas as *strings* inseridas pelo usuário no campo referente ao IP e porta do servidor OPC UA, assim como a lista de *tags*, o intervalo de renovação da seção e o *namespace* respectivos.

Figura 67 – Instanciação da classe OPCUAClass.

```
#region ConexaoServidor
try
{
    myOPCUAServer = new OPCUAClass(InputIp.text, InputPort.text, TagList, false, 30, "4");
}
catch (Opc.Ua.ServiceResultException e)
{
    Debug.Log(e.ToString());
    Debug.Log("Um erro ocorreu ao tentar conectar-se ao servidor OPC.");
}
#endregion
```

Fonte: Autoria própria.

Ao ser instanciada com os argumentos corretos, a classe OPCUAClass os atribui aos seus campos e, ao final, executa o método InitializeOPCUAClient(), visto na Figura 68.

Figura 68 – Construtor da classe OPCUAClass.

```
public OPCUAClass(string serverAddres, string serverport, Dictionary<string, TagClass> taglist,
    bool sessionrenewalRequired, double sessionRenewalMinutes, string nameSpace)
{
    ServerAddress = serverAddres;
    ServerPortNumber = serverport;
    MyApplicationName = "MyApplication";
    TagList = taglist;
    SessionRenewalRequired = sessionrenewalRequired;
    SessionRenewalPeriodMins = sessionRenewalMinutes;
    OPCNameSpace = nameSpace;
    LastTimeOPCServerFoundAlive = DateTime.Now;
    InitializeOPCUAClient();
}
```

Fonte: Autoria própria.

Esse método, por sua vez, constitui-se como a parte principal da execução do cliente e é responsável por realizar todas as configurações relativas tanto à conexão com o servidor, quanto à adição das *tags* a serem monitoradas.

Inicialmente, o *endpoint* de conexão é configurado e, após criar a sessão OPC, uma lista de itens do tipo *MonitoredItem* é criada, onde são adicionados os itens a serem monitorados, correspondentes a cada uma das *tags* definidas pelo usuário. O monitoramento, por sua vez, é possível inscrevendo o *event handler* *OnTagValueChange* nos eventos associados a cada item na lista.

Por fim, são adicionados os itens da lista na inscrição da sessão, a qual é primeiramente adicionada à sessão OPC e, posteriormente, criada, estabelecendo assim a conexão com o servidor, como na Figura 69.

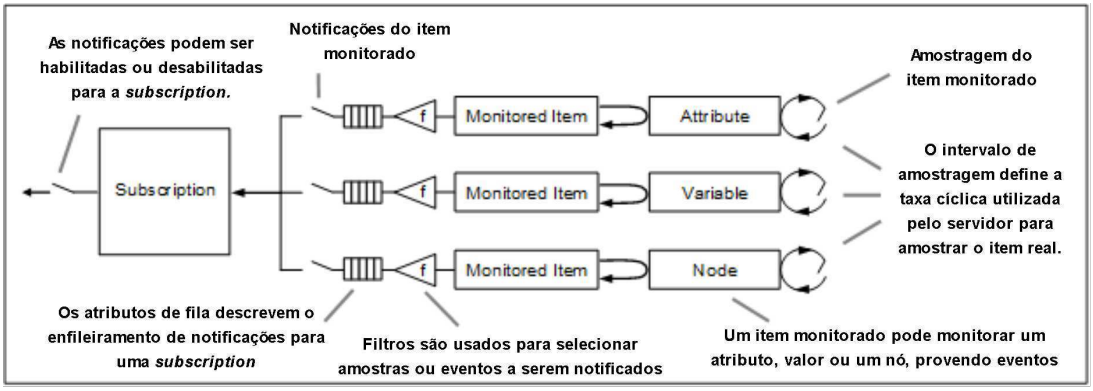
Figura 69 – Criação da sessão OPC e adição dos itens monitorados.

```
public void InitializeOPCUAClient()
{
    /* #region  ConfiguracoesIniciais */…

    string serverAddress = ServerAddress;
    var selectedEndpoint = CoreClientUtils.SelectEndpoint("opc.tcp://" + serverAddress + ":" +
        ServerPortNumber + "", useSecurity: SecurityEnabled, operationTimeout: 15000);

    OPCSession = Session.Create(config,
        new ConfiguredEndpoint(null, selectedEndpoint, EndpointConfiguration.Create(config)),
            false, "", 60000, null, null).GetAwaiter().GetResult();
    {
        var subscription = new Subscription(OPCSession.DefaultSubscription) { PublishingInterval = 1000 };
        var list = new List<MonitoredItem> { };
        list.Add(new MonitoredItem(subscription.DefaultItem) { DisplayName = "ServerStatusCurrentTime",
            StartNodeId = "i=2258" });

        foreach (KeyValuePair<string, TagClass> td in TagList)
        {
            list.Add(new MonitoredItem(subscription.DefaultItem) { DisplayName = td.Value.DisplayName,
                StartNodeId = "ns=" + OPCNameSpace + ";s=" + td.Value.NodeID + "" });
        }

        list.ForEach(i => i.Notification += OnTagValueChange);
        subscription.AddItems(list);
        OPCSession.AddSubscription(subscription);
        subscription.Create();
    }
}
```

Fonte: Autoria própria.

O esquema da classe MonitoredItem, ilustrado por meio de seu modelo na Figura 70, permite identificar a sequência de execução das funções e a sua organização, para a leitura de novos valores nos itens monitorados. Para a presente aplicação, dentre os três tipos de itens monitorados, utilizou-se apenas de itens do tipo *Node*.

Figura 70 – Modelo da classe MonitoredItem.



Fonte: Adaptado de OPC Foundation.

A funcionalidade do botão 3 corresponde à desconexão do cliente com o servidor. Quando esse botão é clicado, o método OPCDisconnect da aplicação, é executado, o qual executa o método Close de OPCSession, responsável por fechar a sessão criada.

Figura 71 – Desconexão do cliente com o servidor OPC UA.

```
public void OPCDisconnect()
{
    try
    {
        myOPCUAServer.OPCSession.Close();
        if (!myOPCUAServer.OPCSession.Connected)
        {
            Debug.Log("Foi possível fechar a sessão OPC.");
        }
    }
    catch (Exception e)
    {
        Debug.Log(e.ToString());
        Debug.Log("Não foi possível fechar a sessão OPC.");
    }
}
```

Fonte: Autoria própria.

A funcionalidade do botão 4 corresponde à leitura das *tags* de saída monitoradas. Quando esse botão é clicado, a variável booleana *readTags* assume o valor verdadeiro, inverso ao inicial (falso) e, após isso, no método Update da aplicação, executado em um *loop* infinito, a leitura das *tags* configuradas é habilitada.

A depender do valor das *tags* (verdadeiro ou falso), a cor do *toggle* correspondente será verde, no primeiro caso, ou vermelho, no segundo, permitindo ao usuário visualizar o estado da *tag* na aplicação. Além disso, o valor da variável de controle correspondente é alterado, o qual é utilizado nos códigos relativos à dinâmica dos componentes 3D.

Figura 72 – Trecho do código de leitura das *tags* na aplicação.

```
if (readTags)
{
    if (TagList["atuadorCilindrico"].CurrentValue == "True")
    {
        if(!meu_atuador_avanco.avancandoOPC)
        {
            ToggleSaidaAvancoAtuadorCilindrico.GetComponentInChildren<Image>().color = Color.green;
        }
        meu_atuador_avanco.recuandoOPC = false;
        meu_atuador_avanco.avancandoOPC = true;
    }

    if (TagList["atuadorCilindrico"].CurrentValue == "False")
    {
        if(!meu_atuador_avanco.recuandoOPC)
        {
            ToggleSaidaAvancoAtuadorCilindrico.GetComponentInChildren<Image>().color = Color.red;
        }
        meu_atuador_avanco.avancandoOPC = false;
        meu_atuador_avanco.recuandoOPC = true;
    }
```

Fonte: Autoria própria.

A atualização dos valores das *tags* ocorre no *event handler* OnTagValueChange, mencionado anteriormente. No código implementado na Figura 73, os novos valores do item monitorado são atribuídos à *tag* correspondente da classe TagList.

Figura 73 – Trecho do código de implementação da leitura das *tags* na dll.

```
public void OnTagValueChange(MonitoredItem item, MonitoredItemNotificationEventArgs e)
{
    foreach (var value in item.DequeueValues())
    {
        if (item.DisplayName == "ServerStatusCurrentTime")
        {
            LastTimeOPCServerFoundAlive = value.SourceTimestamp.ToLocalTime();
        }
        else
        {
            if (TagList.ContainsKey(item.DisplayName))
            {
                if (value.Value != null)
                {
                    TagList[item.DisplayName].LastGoodValue = value.Value.ToString();
                    TagList[item.DisplayName].CurrentValue = value.Value.ToString();
                    TagList[item.DisplayName].LastUpdatedTime = DateTime.Now;
                    TagList[item.DisplayName].LastSourceTimeStamp = value.SourceTimestamp.ToLocalTime();
                    TagList[item.DisplayName].StatusCode = value.StatusCode.ToString();
                }
                else
                {
                    TagList[item.DisplayName].StatusCode = value.StatusCode.ToString();
                    TagList[item.DisplayName].CurrentValue = null;
                }
            }
        }
    }
    InitialisationCompleted = true;
}
```

Fonte: Autoria própria.

A funcionalidade do botão 5, por sua vez, corresponde à escrita de valores nas *tags* de entrada. Quando esse botão é clicado, a variável booleana *writeTags* assume o valor verdadeiro, inverso ao inicial (falso), o qual habilita a escrita de valores nessas *tags*.

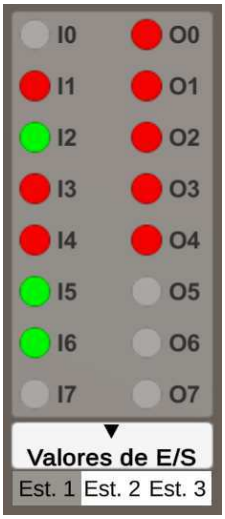
Na aplicação, a escrita de valores nas *tags* de entradas foi implementada atualizando o valor da propriedade isOn de *toggles*, como o visto na Figura 74, correspondentes a cada bit, no mapa de E/S. Essa propriedade refere-se ao valor (verdadeiro ou falso) da *checkbox* desse objeto, o qual se inverte quando esta é clicada.

Figura 74 – *Toggle*.

Fonte: Autoria própria.

Optou-se, então, por criar um painel com *toggles* modificados, visto na Figura 75, de tal forma que essas pudessem ser utilizadas tanto com o objetivo de permitir ao usuário visualizar o valor da *tag* ou variável correspondente, assim como, disparar um evento, a partir da mudança do valor da variável respectiva na aplicação.

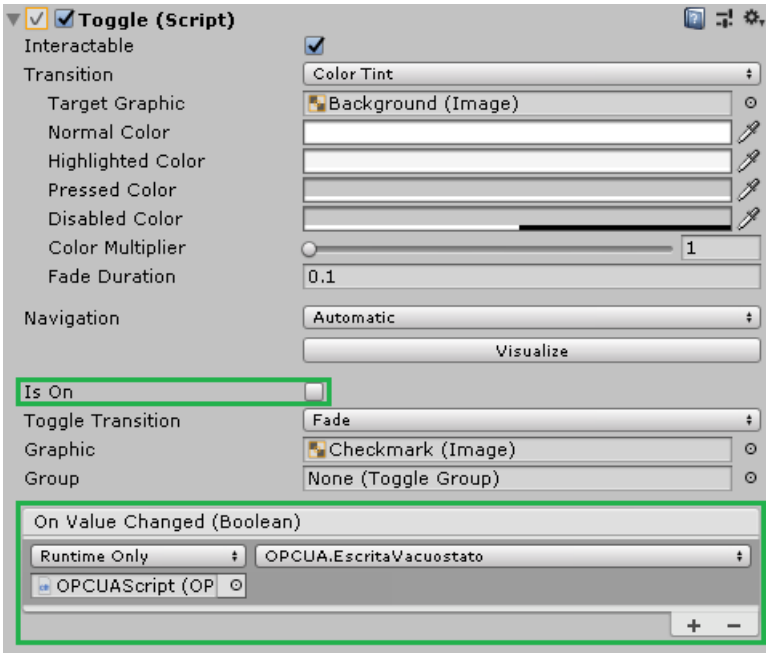
Figura 75 – Painel com *toggles* modificados.



Fonte: Autoria própria.

O componente *Toggle*, do tipo *script*, vinculado ao *gameobject Toggle* permite, portanto, associar uma ação ao evento mencionado anteriormente, como na Figura 76. Nesse caso, associou-se a execução do método EscritaVacuostato(), do *script* OPCUA.

Figura 76 – Associação de uma ação ao evento de mudança de valor do *toggle*.



Fonte: Autoria própria.

O método EscritaVacuostato na Figura 77, por sua vez, escreve o valor verdadeiro ou falso na *tag* correspondente, por meio do método EscreveNasTags, que recebe o *nodeId* e o valor da propriedade isOn do *toggle* como argumentos. Dessa forma, a escrita de valores nas *tags* ocorre só quando há uma troca do valor dessas, diminuindo a sobrecarga da aplicação quando conectada ao servidor OPC UA.

Figura 77 – Método EscritaVacuostato.

```
public void EscritaVacuostato()
{
    string rep = "ns=4;s=|var|CODESYS Control Win V3.Application.";
    if(writeTags)
    {
        if (ToggleVacuostato.isOn)
        {
            myOPCUAServer.EscreveNasTags(rep + InputDistribuicaoEntradaBit3.text, true);
        }

        if (!ToggleVacuostato.isOn)
        {
            myOPCUAServer.EscreveNasTags(rep + InputDistribuicaoEntradaBit3.text, false);
        }
    }
}
```

Fonte: Autoria própria.

Finalmente, no método EscreveNasTags da Figura 78 implementado no projeto do cliente, inicialmente realiza-se as configurações do tipo e valor da variável valueToWrite, do tipo WriteValue e, em seguida, utiliza-se dessa variável para escrever o valor recebido como argumento na *tag* correspondente, por meio do método Write.

Figura 78 – Trecho do código de implementação da escrita nas *tags* na dll.

```
public void EscreveNasTags(string idNode, bool novoValor)
{
    WriteValue valueToWrite = new WriteValue();
    DataValue teste = new DataValue();
    teste.Value = false;

    valueToWrite.NodeId = idNode;
    valueToWrite.AttributeId = 13;
    valueToWrite.Value.Value = ChangeType(teste, novoValor);
    valueToWrite.Value.StatusCode = StatusCodes.Good;
    valueToWrite.Value.ServerTimestamp = DateTime.MinValue;
    valueToWrite.Value.SourceTimestamp = DateTime.MinValue;

    WriteValueCollection valuesToWrite = new WriteValueCollection();
    valuesToWrite.Add(valueToWrite);
    StatusCodeCollection results = null;
    DiagnosticInfoCollection diagnosticInfos = null;

    if (novoValor2 != null)
    {
        try{OPCSession.Write(null, valuesToWrite, out results, out diagnosticInfos);}
        /* #region  catch */ ...
        /* #region  Validação da resposta */ ...
    }
}
```

Fonte: Autoria própria.

4 Resultados

Como resultado da aplicação da metodologia apresentada no capítulo anterior, foi criado um gêmeo digital do MPS da Festo, inserido em uma aplicação que permite testar a implementação de projetos de automação industrial utilizando o padrão de comunicação OPC UA. Em síntese, a aplicação multiplataforma comporta-se como um ambiente por meio do qual é possível controlar o gêmeo digital e visualizar a sua dinâmica sem ou com recursos de realidade aumentada.

4.1 Interface com o Usuário

Na primeira execução da aplicação é exibida a interface "Painel Inicial", mostrada na Figura 79, que introduz o usuário com um resumo acerca do que se trata a aplicação, assim como o seu propósito. Além do resumo, o usuário pode sair da aplicação ou avançar para o próximo painel clicando nos botões "Sair" ou "Iniciar", respectivamente.

Figura 79 – Interface "Painel Inicial".

Fonte: Autoria própria.

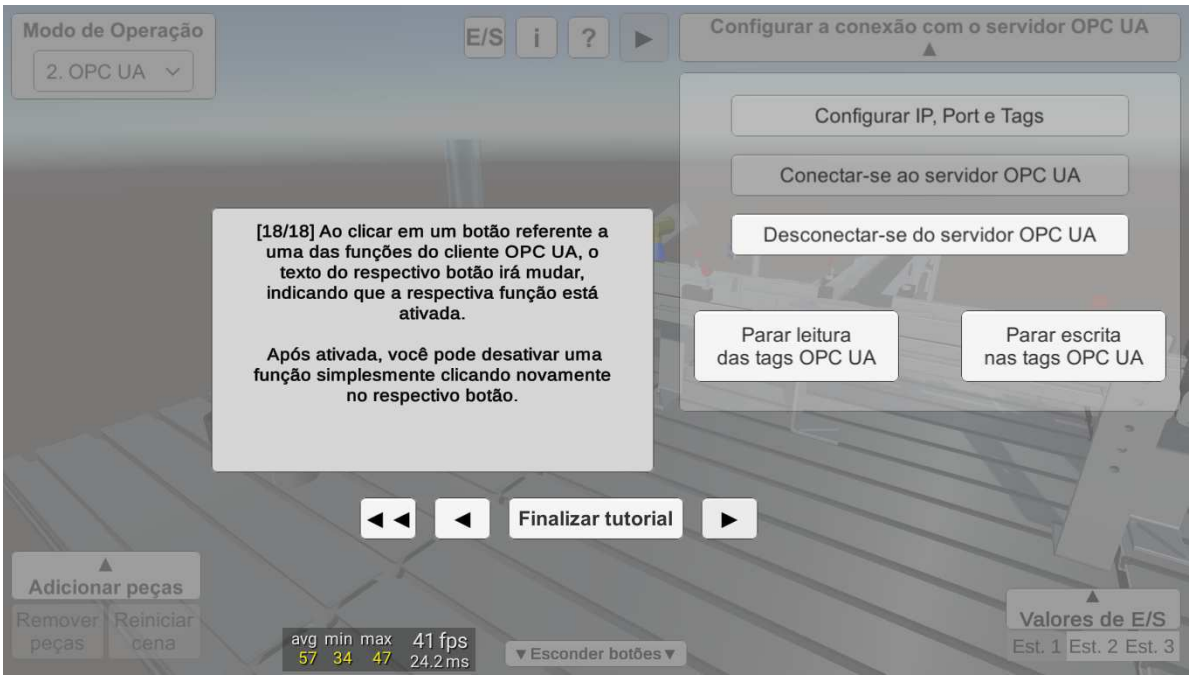
Após clicar no botão "Iniciar", caso seja esta a primeira execução da aplicação, o usuário é direcionado para o painel seguinte, no qual lhe é apresentado o tutorial constituído de 18 partes, com um resumo das funcionalidades de cada componente de interface com o usuário presente na aplicação. Nesse tutorial, o usuário é instruído desde como selecionar o modo de operação do gêmeo digital (Figura 80) até como acionar as funções do cliente OPC UA na aplicação (Figura 81).

Figura 80 – Tutorial: escolha do modo de operação.

Fonte: Autoria própria.

O usuário pode navegar entre os passos do tutorial por meio dos quatro botões "<<", "<", "Finalizar tutorial", ">" e ">>", abaixo do painel central, que permitem, respectivamente, voltar para o primeiro passo, retornar um passo, fechar o tutorial, avançar um passo e ir para o último passo. Os botões são exibidos a depender do passo em que o usuário se encontra e o tutorial pode ser novamente acessado clicando no botão "?".

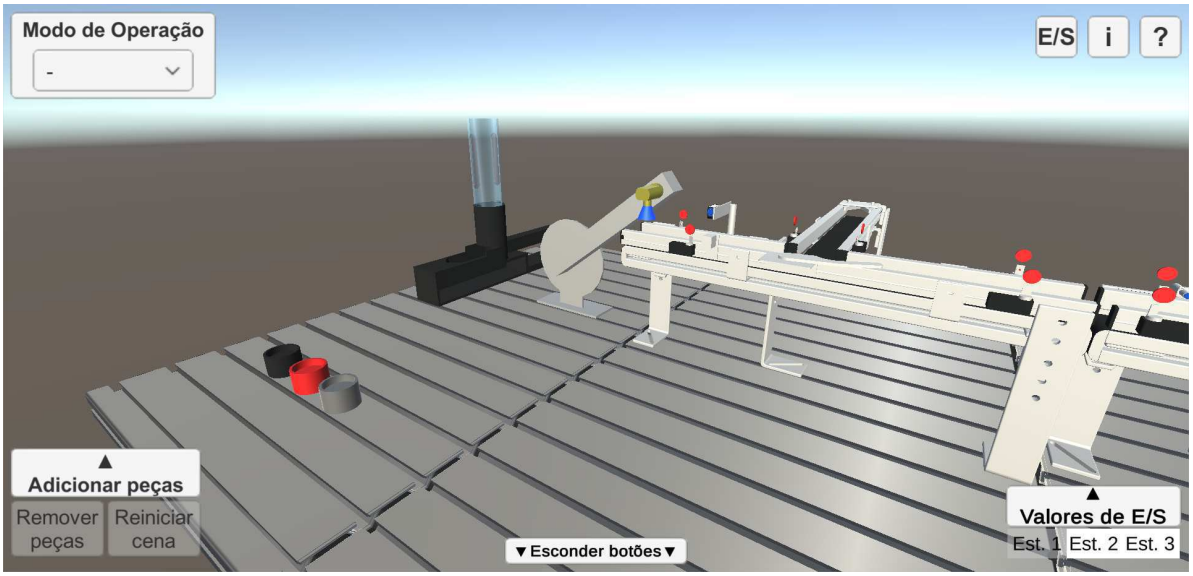
Figura 81 – Tutorial: acionamento das funções do cliente OPC UA.



Fonte: Autoria própria.

Ao fechar esse tutorial ou ao executar a aplicação pela segunda vez, tendo já finalizado o tutorial uma vez, o usuário é direcionado, portanto, para a tela principal da aplicação (Figura 82), na qual tem acesso a todas as funcionalidades dessa.

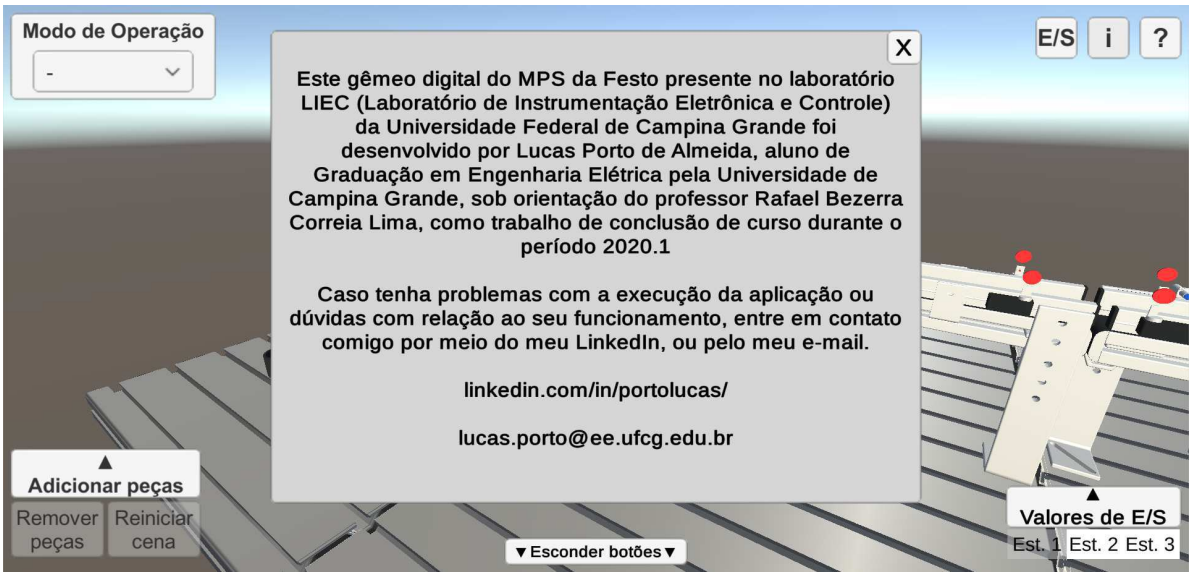
Figura 82 – Tela principal.



Fonte: Autoria própria.

Ao clicar no botão "i", no canto superior da tela principal, é exibido o painel "Sobre" (Figura 83), por meio do qual o usuário é informado sobre o autor da aplicação, o seu contexto de desenvolvimento, o professor orientador, além de contatos para tirar dúvidas ou enviar comentários. Ao clicar no botão "X", o usuário retorna à tela principal.

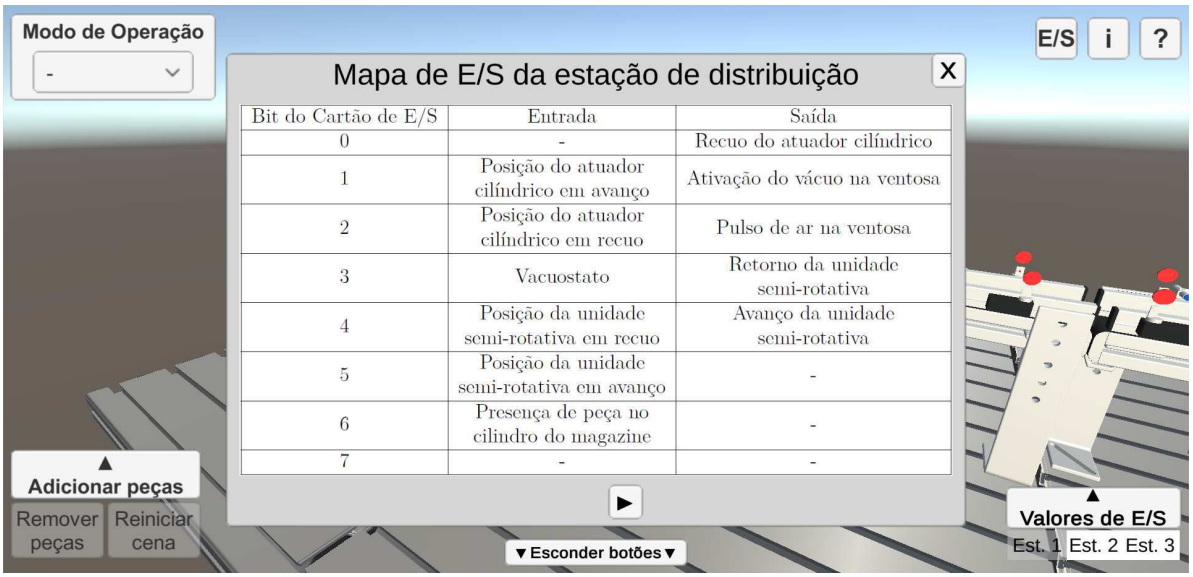
Figura 83 – Painel "Sobre".



Fonte: Autoria própria.

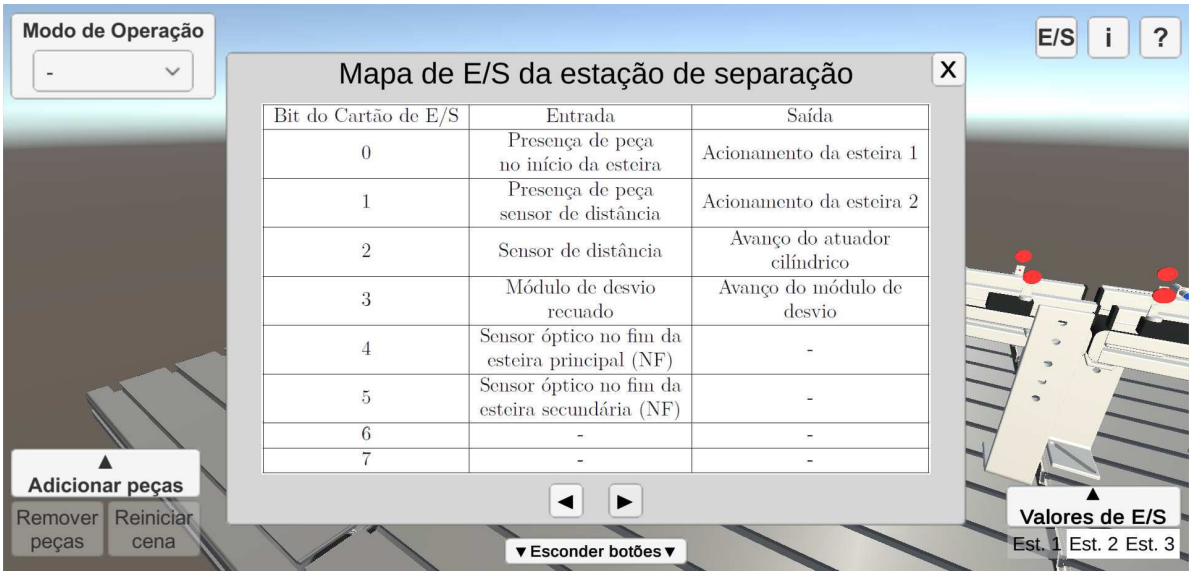
Ao clicar no botão "E/S", no canto superior da tela principal, é exibido o painel "Mapas de E/S", por meio do qual o usuário pode visualizar tabelas referentes aos mapas dos bits de entrada e saída para as estações de distribuição, separação e classificação (Figuras 84, 85 e 86, respectivamente). O usuário pode navegar entre os três mapas por meio dos botões "<" e ">", que permitem, respectivamente, voltar para a estação anterior e avançar para a próxima estação, e são exibidos a depender do mapa acessado.

Figura 84 – Painel "Mapas de E/S": estação de distribuição.



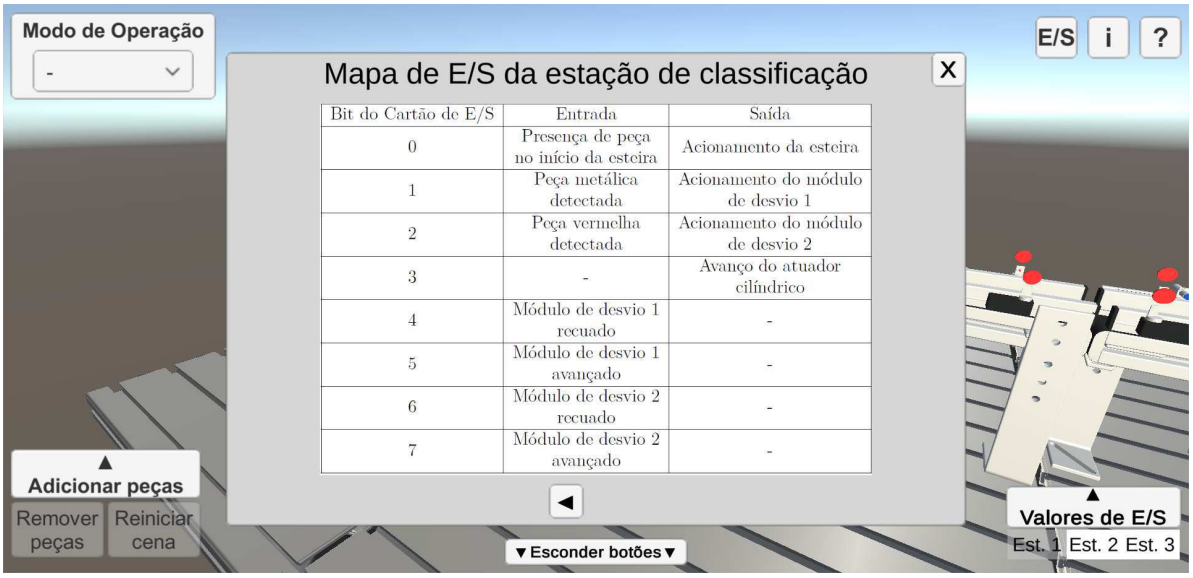
Fonte: Autoria própria.

Figura 85 – Painel "Mapas de E/S": estação de separação.



Fonte: Autoria própria.

Figura 86 – Painel "Mapas de E/S": estação de classificação.

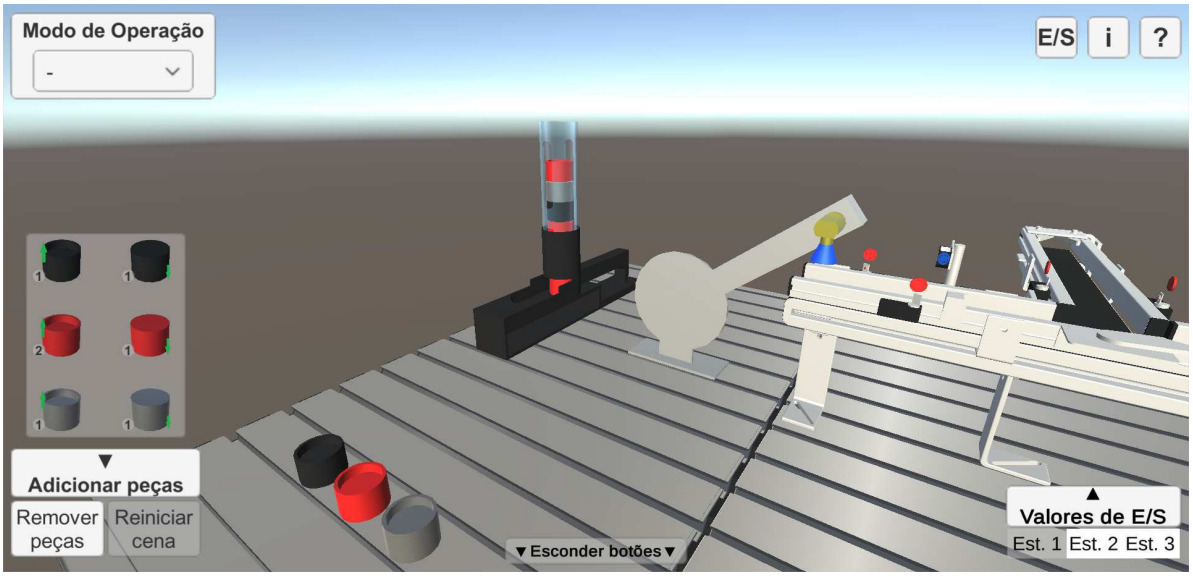


Fonte: Autoria própria.

Ao clicar no botão "Adicionar peças", no canto inferior esquerdo da tela principal, é exibido, acima desse botão, o painel "Adição de peças" (Figura 87), no qual há 6 botões que permitem ao usuário inserir peças pretas, vermelhas ou metálicas, com orientação para cima ou para baixo, no cilindro do magazine.

Caso haja oito peças no seu interior, os botões são desabilitados, sendo habilitados novamente quando essa quantidade for menor que oito. O botão "Remover peças", por sua vez, permite deletar as peças inseridas no cilindro antes de iniciar a execução.

Figura 87 – Painel "Adição de peças".



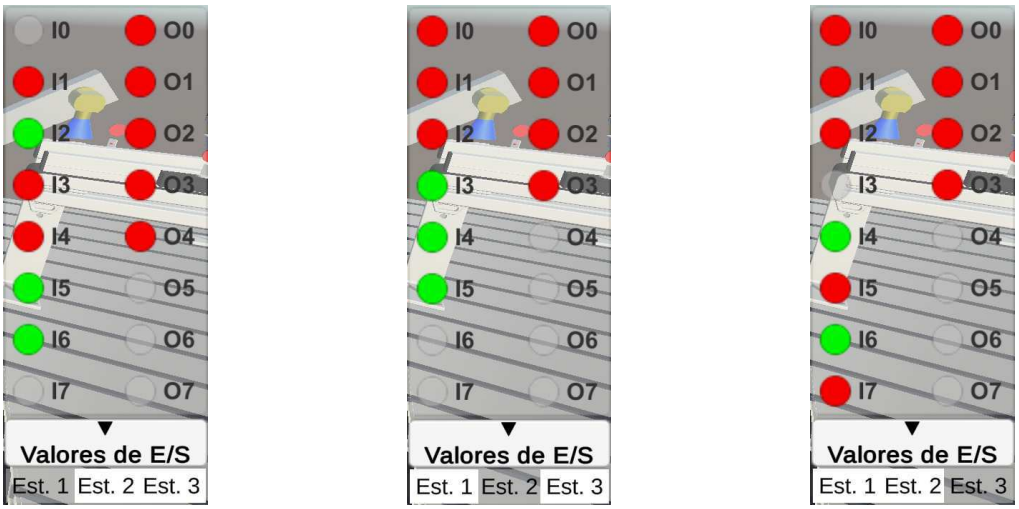
Fonte: Autoria própria.

No canto inferior direito da tela principal, ao clicar no botão "Valores de E/S", é exibido, acima desse botão, o painel de mesmo nome, no qual há 16 *toggles* modificados para cada estação (Figuras 88a, 88b e 88c), totalizando 48, que funcionam como indicadores de valor. Esses indicadores permitem ao usuário verificar os estados das variáveis, na aplicação, associadas aos bits de entrada e saída nos mapas, apresentados anteriormente.

Os conjuntos de E/S para cada estação são acessados ao clicar nos botões "Est. 1, 2 e 3", referentes respectivamente às estações de distribuição, separação e classificação, e os *toggles* de cor cinza, por sua vez, correspondem aos bits que não foram mapeados.

Figura 88 – Painel "Valores de E/S": valores iniciais de E/S de cada estação.

(a) Distribuição. (b) Separação. (c) Classificação.



Fonte: Autoria própria.

Ao selecionar a opção "2. OPC UA" no menu suspenso "Modo de Operação" (Figura 89), no canto superior esquerdo da tela principal, são exibidos ao usuário, no canto superior direito, os botões "<" e "Config. OPC UA" (Figura 90), estando o último desabilitado.

Ao clicar no botão "<", esse é trocado pelo botão ">", e o botão "Config. OPC UA" é trocado pelo botão "Configurar a conexão com o servidor OPC UA" (Figura 91), o que permite ao usuário uma melhor visualização da região superior da tela principal.

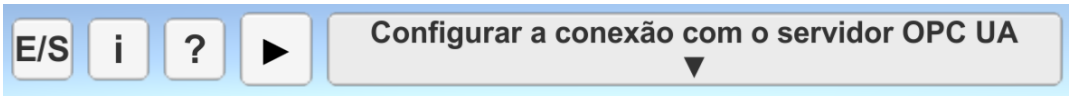
Figura 89 – Menu suspenso "Modo de Operação": seleção da opção "2. OPC UA".

Fonte: Autoria própria.

Figura 90 – Botões "<" e "Config. OPC UA".

Fonte: Autoria própria.

Figura 91 – Botões ">" e "Configurar conexão com o servidor OPC UA".

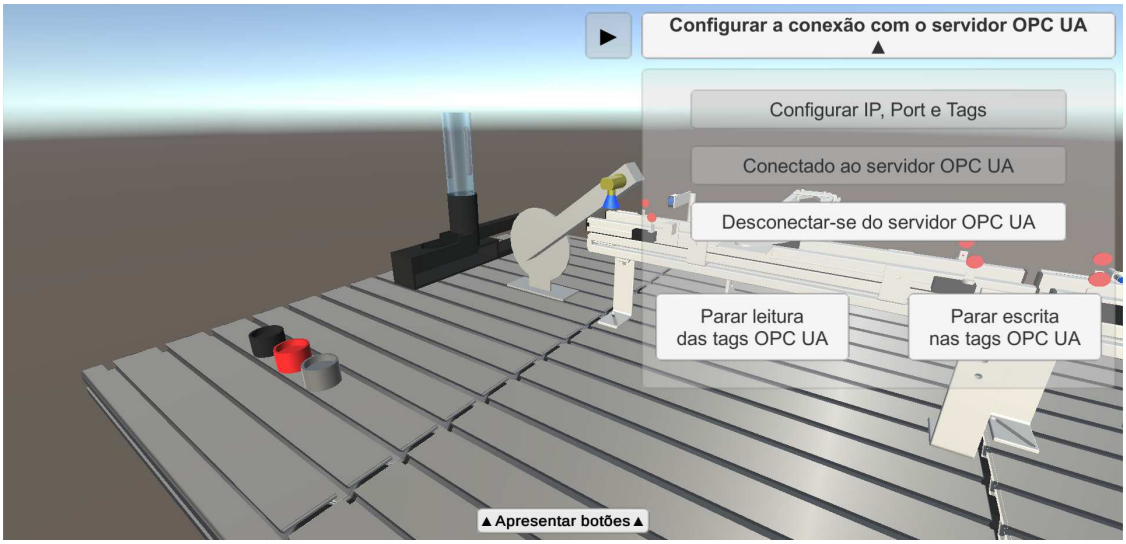


Fonte: Autoria própria.

Caso nenhum painel esteja sendo exibido, o botão "Esconder botões", no canto inferior da tela (Figura 92), é habilitado. Ao clicar nesse botão, o seu texto é trocado por "Apresentar botões" e todos os botões da tela principal, assim como o menu suspenso são ocultados, com exceção dos referentes à conexão com o servidor OPC UA.

Ao clicar novamente nesse botão, o seu texto retorna ao valor original e todos os componentes de interface com o usuário voltam a ser exibidos na tela principal. Essa funcionalidade foi implementada a fim de ampliar, ao usuário, a visibilidade dos componentes do sistema na aplicação.

Figura 92 – Botão "Esconder botões": botões e menu suspenso ocultados.



Fonte: Autoria própria.

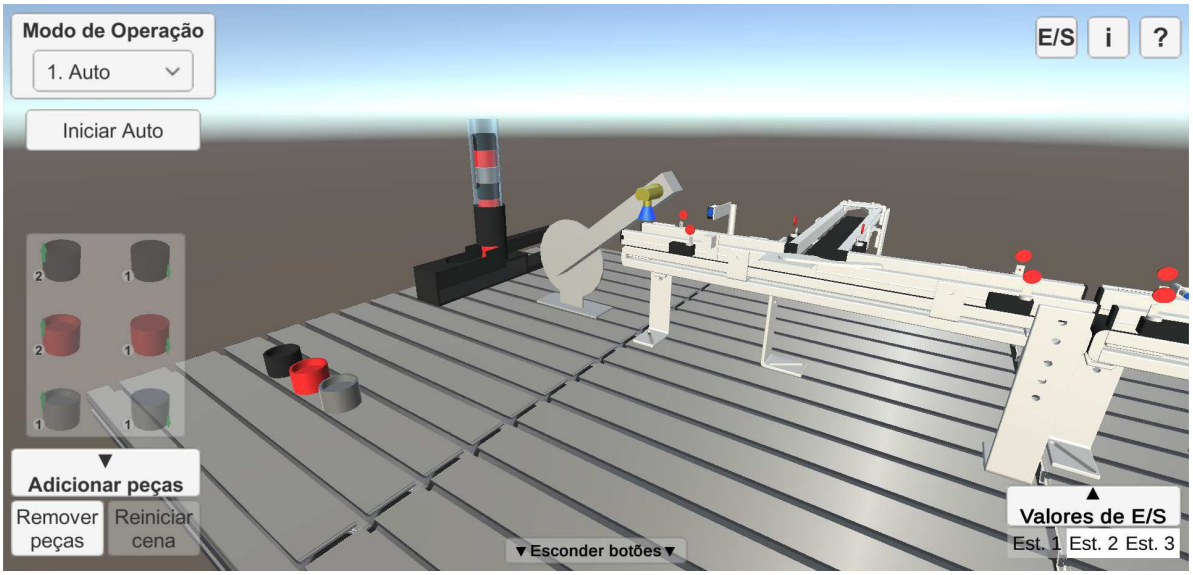
4.2 Modo de Operação Automático

Ao selecionar a opção "1. Auto" no menu suspenso "Modo de Operação", logo abaixo desse menu, o botão desabilitado "Iniciar Auto" é exibido ao usuário, o qual é

habilitado a partir do momento em que uma peça é inserida no cilindro do magazine (Figura 93).

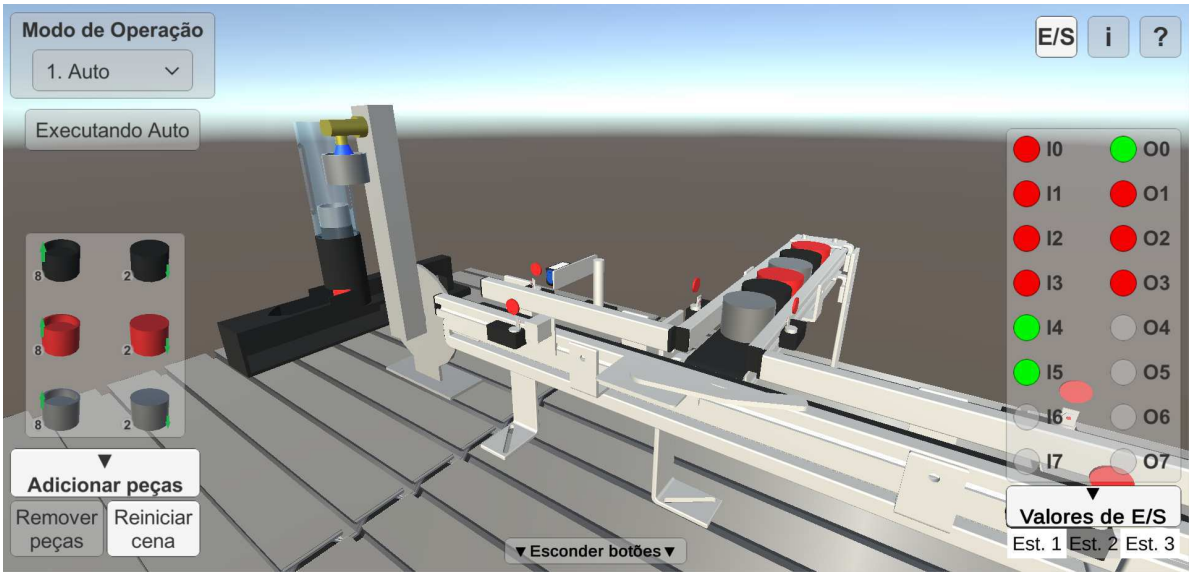
Ao clicar nesse botão, o seu texto será trocado para "Executando Auto" e a aplicação executará uma demonstração do funcionamento de todas as 3 estações. Essa demonstração possibilita ao usuário uma compreensão inicial do funcionamento dos componentes do sistema, de maneira a auxiliá-lo na implementação do projeto para o modo de operação conectado ao servidor OPC UA.

Figura 93 – Modo de operação automático: botão "Iniciar Auto" habilitado.



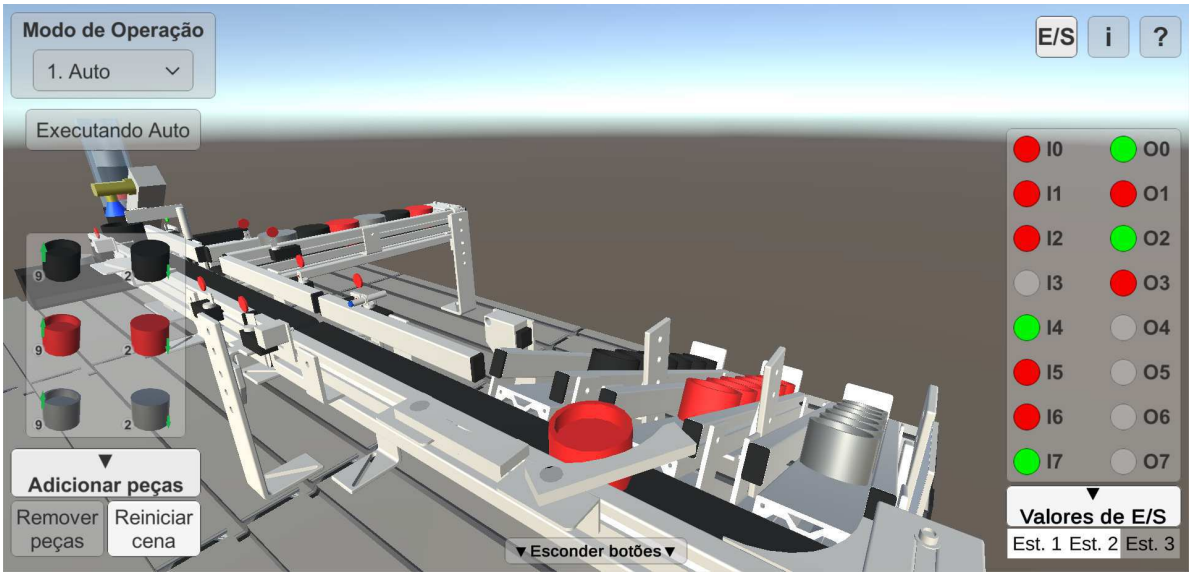
Fonte: Autoria própria.

Figura 94 – Modo de operação automático: estação de separação.



Fonte: Autoria própria.

Figura 95 – Modo de operação automático: estação de classificação.

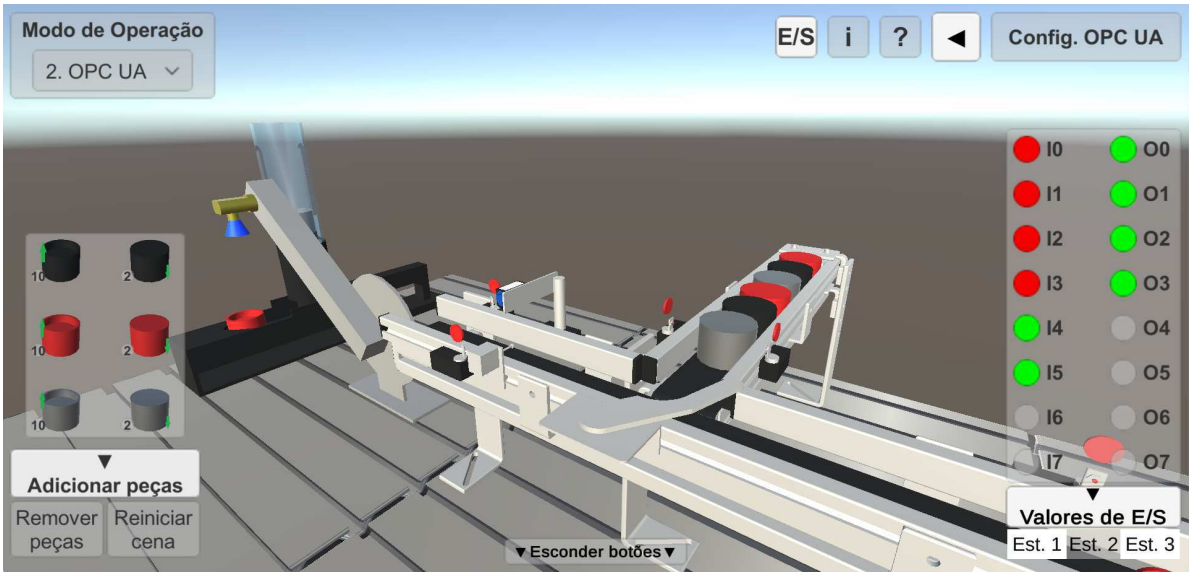


Fonte: Autoria própria.

4.3 Modo de Operação conectado a um servidor OPC UA

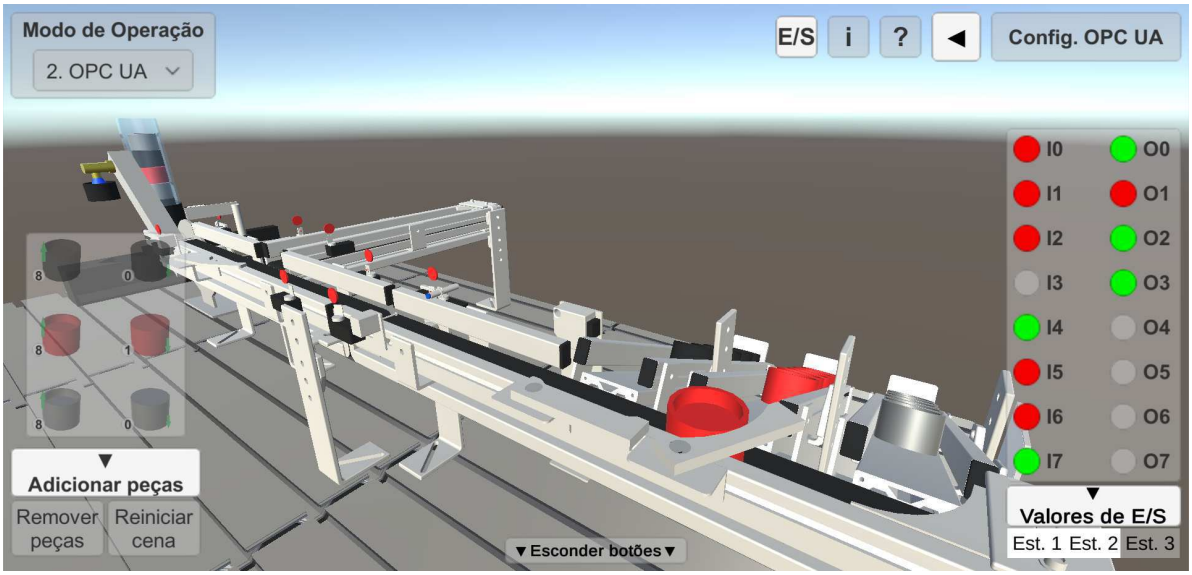
Ao selecionar a opção "2. OPC UA" no menu suspenso "Modo de Operação", a aplicação irá exibir e habilitar os botões referentes à conexão com o servidor OPC UA. Feitas as configurações e iniciada a conexão, o usuário pode iniciar as funções do cliente OPC UA e controlar os componentes da aplicação, de forma a alcançar o mesmo objetivo funcional observado anteriormente.

Figura 96 – Modo de operação conectado a um servidor OPC UA: estação de separação.



Fonte: Autoria própria.

Figura 97 – Modo de operação conectado a um servidor OPC UA: estação de classificação.

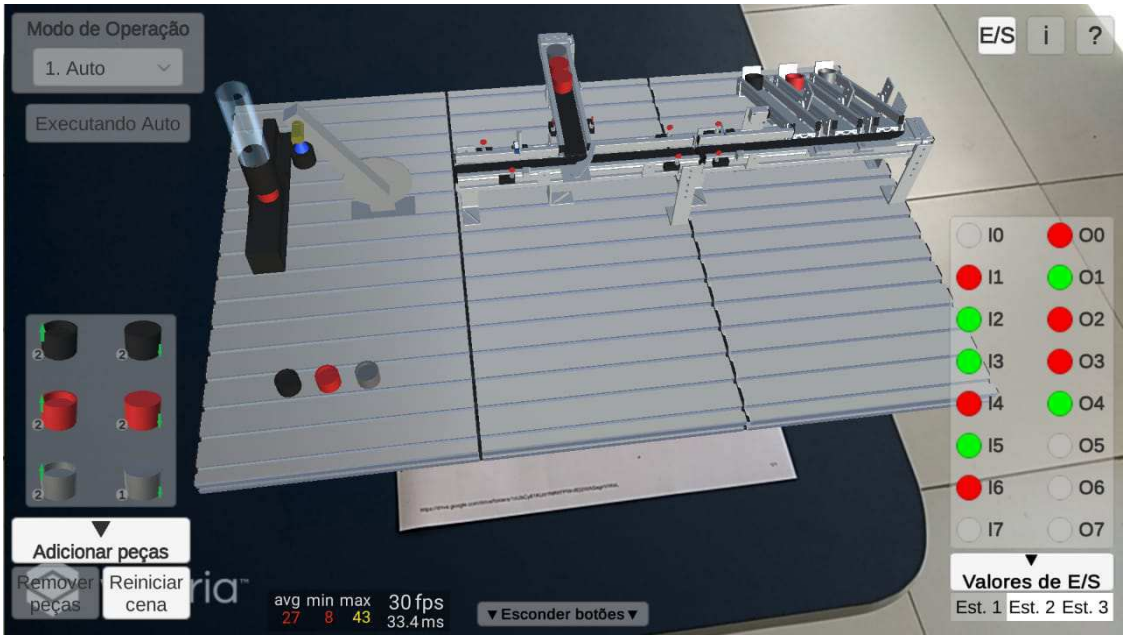


Fonte: Autoria própria.

4.4 Operação com recursos de realidade aumentada

Para os dispositivos móveis do tipo *smartphones* com sistema operacional Android, criou-se também, além da versão padrão, uma versão à parte, com recursos de realidade aumentada. É possível ao usuário, portanto, executar a aplicação da mesma forma que faria para a versão padrão, porém visualizando os componentes do sistema em 3D com realidade aumentada (Figura 98).

Figura 98 – Aplicação na versão com recursos de realidade aumentada.



Fonte: Autoria própria.

5 Conclusões

Ao final deste trabalho, foi desenvolvido um gêmeo digital do MPS da Festo, presente no Laboratório de Instrumentação Eletrônica e Controle da UFCG, o qual foi inserido em uma aplicação construída para as plataformas *desktop* Windows e Linux e para a plataforma *mobile* Android, incluindo uma versão adicional para esta, com recursos integrados de realidade aumentada.

A aplicação desenvolvida permite ao usuário visualizar uma demonstração da funcionalidade das três estações modulares constituintes do sistema, assim como implementar o seu controle a partir de um servidor OPC UA. Neste sentido, a aplicação consiste numa ferramenta possível de ser utilizada para diversos propósitos, dentre esses, a otimização do funcionamento do sistema físico, assim como no contexto educacional, como uma ferramenta auxiliar de ensino, inclusive na modalidade remota.

De forma geral, as respostas obtidas com relação à experiência de uso com a aplicação foram positivas, de forma que sugerem fortemente a sua viabilidade como ferramenta de ensino no ambiente da graduação.

Considera-se portanto, diante da proposta inicial, que os objetivos estabelecidos foram alcançados. Tendo sido, dentre esses, a modelagem 3D dos componentes do sistema o maior desafio observado.

Por fim, espera-se que os passos apresentados neste trabalho, assim como o seu resultado final, motivem outras implementações dos conceitos abordados tanto com o propósito educacional, quanto para diversas outras áreas da engenharia, que podem se beneficiar de todo o seu potencial.

5.1 Trabalhos Futuros

Considerando a vasta gama de funcionalidades das ferramentas utilizadas, é possível aprimorar a aplicação final em diferentes aspectos, sem comprometer a sua performance nos dispositivos alvo. Dentre esses, destacam-se os aspectos relacionados ao nível de fidelidade gráfica componentes 3D do sistema, assim como a usabilidade dos componentes de interface com o usuário da aplicação e as funcionalidades disponibilizadas nesta.

De forma específica, pode-se listar as seguintes sugestões de aprimoramento a serem consideradas para as possíveis novas versões da aplicação:

- Integração dos demais componentes elétricos e pneumáticos do sistema ao gêmeo digital, de forma a aumentar o seu nível de espelhamento;

- Aprimoramento dos detalhes gráficos dos componentes 3D modelados, com relação ao seu formato, escala e material;
- Melhoria da disposição e tamanho dos componentes de interface com o usuário na aplicação, de maneira a facilitar o seu uso;
- Otimização das funcionalidades relativas ao rastreamento do código QR, assim como o posicionamento e estabilidade dos objetos 3D exibidos em RA sobre este.

Referências

AUTOCAD. *Sobre como importar e exportar arquivos FBX*. Disponível em: <https://autode.sk/2QJHDA0>. Acessado em 31-08-2020. Citado na página 25.

AZUMA, R. T. A survey of augmented reality. *Presence: Teleoperators & Virtual Environments*, MIT Press, v. 6, n. 4, p. 355–385, 1997. Citado na página 8.

BOLTON, R. N. et al. Customer experience challenges: bringing together digital, physical and social realms. *Journal of Service Management*, Emerald Publishing Limited, 2018. Citado na página 6.

BUCSAI, S. et al. Control and monitoring of iot devices using mixed reality developed by unity engine. In: . [S.l.: s.n.], 2020. p. 1–8. Citado na página 2.

CARDOSO, L. F. de S.; MARIANO, F. C. M. Q.; ZORZAL, E. R. A survey of industrial augmented reality. *Computers & Industrial Engineering*, Elsevier, v. 139, p. 106159, 2020. Citado na página 8.

COMMISSION, N. I. et al. Data for the public good. *NIC Report*, 2017. Citado na página 6.

EBEL, F.; PANY, M. *Distributing station manual*. 1. ed. Denkendorf: Festo Didactic GmbH Co. KG, 2006. Citado 5 vezes nas páginas 3, 15, 16, 17 e 18.

EBEL, F.; PANY, M. *Separating station manual*. 1. ed. Denkendorf: Festo Didactic GmbH Co. KG, 2006. Citado 2 vezes nas páginas 20 e 21.

EBEL, F.; PANY, M. *Sorting station manual*. 1. ed. Denkendorf: Festo Didactic GmbH Co. KG, 2006. Citado 2 vezes nas páginas 23 e 24.

FOUNDATION, O. *Unified Architecture*. 2020. Disponível em: <https://opcfoundation.org/about/opc-technologies/opc-ua/>. Acessado em 02-12-2020. Citado na página 10.

GLAESSGEN, E.; STARGEL, D. The digital twin paradigm for future nasa and us air force vehicles. In: *53rd AIAA/ASME/ASCE/AHS/ASC structures, structural dynamics and materials conference 20th AIAA/ASME/AHS adaptive structures conference 14th AIAA*. [S.l.: s.n.], 2012. p. 1818. Citado na página 6.

GRIEVES, M. Digital twin: Manufacturing excellence through virtual factory replication. 03 2015. Citado na página 5.

GRIEVES, M. Origins of the digital twin concept. p. 7, 08 2016. Citado na página 1.

GRIEVES, M.; VICKERS, J. Digital twin: Mitigating unpredictable, undesirable emergent behavior in complex systems. In: *Transdisciplinary perspectives on complex systems*. [S.l.]: Springer, 2017. p. 85–113. Citado na página 6.

KRITZINGER, W. et al. Digital twin in manufacturing: A categorical literature review and classification. *IFAC-PapersOnLine*, Elsevier, v. 51, n. 11, p. 1016–1022, 2018. Citado na página 7.

MAHNKE, W.; LEITNER, S.-H.; DAMM, M. *OPC unified architecture*. [S.l.]: Springer Science & Business Media, 2009. Citado na página 9.

MARTIN, J.; BOHUSLAVA, J. Augmented reality as an instrument for teaching industrial automation. In: IEEE. *2018 Cybernetics & Informatics (K&I)*. [S.l.], 2018. p. 1–5. Citado na página 8.

MEDIA, V. *With new realities to build, Unity positioned to become tech giant*. 2017. Disponível em: <https://techcrunch.com/2017/05/25/with-new-realities-to-build-unity-positioned-to-become-tech-giant/>. Acessado em 03-12-2020. Citado na página 11.

MICROSOFT. *.NET Standard*. 2020. Disponível em: <https://docs.microsoft.com/en-us/dotnet/standard/net-standard>. Acessado em 10-11-2020. Citado na página 50.

NEE, A. Y.; ONG, S.-K. Virtual and augmented reality applications in manufacturing. *IFAC proceedings volumes*, Elsevier, v. 46, n. 9, p. 15–26, 2013. Citado na página 8.

PTC. *Getting Started with Vuforia Engine in Unity*. 2020. Disponível em: <https://library.vuforia.com/articles/Training/getting-started-with-vuforia-in-unity.html>. Acessado em 03-11-2020. Citado na página 43.

SÖDERBERG, R. et al. Toward a digital twin for real-time geometry assurance in individualized production. *CIRP Annals*, Elsevier, v. 66, n. 1, p. 137–140, 2017. Citado na página 6.

TAO, F. et al. Digital twin-driven product design, manufacturing and service with big data. *The International Journal of Advanced Manufacturing Technology*, Springer, v. 94, n. 9-12, p. 3563–3576, 2018. Citado na página 6.

UNITY. *Unity UI: Unity User Interface*. Disponível em: <https://docs.unity3d.com/Packages/com.unity.ugui@1.0/manual/index.html>. Acessado em 03-11-2020. Citado na página 38.

UNITY. *Vuforia Engine AR*. Disponível em: <https://docs.unity3d.com/Manual/com.ptc.vuforia.engine.html>. Acessado em 03-11-2020. Citado na página 43.

UNITY. *Supported Model file formats*. 2018. Disponível em: <https://docs.unity3d.com/Manual/3D-formats.html>. Acessado em 31-08-2020. Citado na página 25.

UNITY. *.NET profile support*. 2019. Disponível em: <https://docs.unity3d.com/2019.1/Documentation/Manual/dotnetProfileSupport.html>. Acessado em 10-11-2020. Citado na página 50.

UNITY. *Overview of .NET in Unity*. 2020. Disponível em: <https://docs.unity3d.com/Manual/overview-of-dot-net-in-unity.html>. Acessado em 10-11-2020. Citado na página 50.