David Benetti

Integração entre soluções inspiradas em planejamento automático e ambientes didáticos práticos baseados em CLP

Goiânia
2024

UNIVERSIDADE FEDERAL DE GOIÁS
ESCOLA DE ENGENHARIA ELÉTRICA, MECÂNICA E DE COMPUTAÇÃO

TERMO DE CIÊNCIA E DE AUTORIZAÇÃO PARA DISPONIBILIZAR VERSÕES ELETRÔNICAS DE TRABALHO DE CONCLUSÃO DE CURSO DE GRADUAÇÃO NO REPOSITÓRIO INSTITUCIONAL DA UFG

Na qualidade de titular dos direitos de autor, autorizo a Universidade Federal de Goiás (UFG) a disponibilizar, gratuitamente, por meio do Repositório Institucional (RI/UFG), regulamentado pela Resolução CEPEC no 1240/2014, sem ressarcimento dos direitos autorais, de acordo com a Lei no 9.610/98, o documento conforme permissões assinaladas abaixo, para fins de leitura, impressão e/ou download, a título de divulgação da produção científica brasileira, a partir desta data.

O conteúdo dos Trabalhos de Conclusão dos Cursos de Graduação disponibilizado no RI/UFG é de responsabilidade exclusiva dos autores. Ao encaminhar(em) o produto final, o(s) autor(a)(es)(as) e o(a) orientador(a) firmam o compromisso de que o trabalho não contém nenhuma violação de quaisquer direitos autorais ou outro direito de terceiros.

1. Identificação do Trabalho de Conclusão de Curso de Graduação (TCCG)

Nome(s) completo(s) do(a)(s) autor(a)(es)(as): David Benetti

Título do trabalho: Integração entre soluções inspiradas em planejamento automático e ambientes didáticos práticos baseados em CLP

2. Informações de acesso ao documento (este campo deve ser preenchido pelo orientador) Concorda com a liberação total do documento [ ] SIM [X] NÃO[1]

[1] Neste caso o documento será embargado por até um ano a partir da data de defesa. Após esse período, a possível disponibilização ocorrerá apenas mediante: a) consulta ao(à)(s) autor(a)(es)(as) e ao(à) orientador(a); b) novo Termo de Ciência e de Autorização (TECA) assinado e inserido no arquivo do TCCG. O documento não será disponibilizado durante o período de embargo.

Casos de embargo:

- Solicitação de registro de patente;

- Submissão de artigo em revista científica;

- Publicação como capítulo de livro.

Obs.: Este termo deve ser assinado no SEI pelo orientador e pelo autor.

Documento assinado eletronicamente por **Joao Paulo Da Silva Fonseca**, **Professor do Magistério Superior**, em 25/07/2024, às 09:09, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **David Benetti**, **Discente**, em 06/08/2024, às 09:37, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

A autenticidade deste documento pode ser conferida no site https://sei.ufg.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **4695387** e o código CRC **74E2FF94**.

David Benetti

Integração entre soluções inspiradas em planejamento automático e ambientes didáticos práticos baseados em CLP

Trabalho de conclusão de curso apresentado ao curso de Engenharia de computação, da Escola de Engenharia Elétrica, Mecânica e de Computação, da Universidade Federal de Goiás (UFG), como requisito para obtenção do título de Engenheiro de Computação.

Universidade Federal de Goiás

Escola de Engenharia Elétrica, Mecânica e de Computação

Prof. Orientador: Dr. João Paulo da Silva Fonseca

Goiânia
2024

Ficha de identificação da obra elaborada pelo autor, através do
Programa de Geração Automática do Sistema de Bibliotecas da UFG.

Benetti, David
Integração entre soluções inspiradas em planejamento automático e ambientes didáticos práticos baseados em CLP [manuscrito] / David Benetti. - 2024.
XVI, 16 f.

Orientador: Prof. Dr. João Paulo da Silva Fonseca.
Trabalho de Conclusão de Curso (Graduação) - Universidade Federal de Goiás, Escola de Engenharia Elétrica, Mecânica e de Computação (EMC), Engenharia da Computação, Goiânia, 2024.
Bibliografia. Apêndice.

1. CLP. 2. Indústria 4.0. 3. Integração de sistemas. 4. Planejador automático. 5. Python. I. Fonseca, João Paulo da Silva, orient. II. Título.

UNIVERSIDADE FEDERAL DE GOIÁS
ESCOLA DE ENGENHARIA ELÉTRICA, MECÂNICA E DE COMPUTAÇÃO

ATA DE DEFESA DE TRABALHO DE CONCLUSÃO DE CURSO

Ao(s) vinte e quatro dia(s) do mês de julho do ano de 2024 iniciou-se a sessão pública de defesa do Trabalho de Conclusão de Curso (TCC) intitulado “Integração entre soluções inspiradas em planejamento automático e ambientes didáticos práticos baseados em CLP”, de autoria de David Benetti, do curso de Engenharia de Computação, do(a) Escola de Engenharia Elétrica, Mecânica e de Computação da UFG. Os trabalhos foram instalados pelo(a) Prof. João Paulo da Silva Fonseca - orientador (EMC/UFG) com a participação dos demais membros da Banca Examinadora: Marco Antonio Assfalk de Oliveira (EMC/UFG) e Weslley da Silva Alves (Faculdade SENAI IB). Após a apresentação, a banca examinadora realizou a arguição do(a) estudante. Posteriormente, de forma reservada, a Banca Examinadora atribuiu a nota final de 7,3, tendo sido o TCC considerado aprovado.

Proclamados os resultados, os trabalhos foram encerrados e, para constar, lavrou-se a presente ata que segue assinada pelos Membros da Banca Examinadora.

Documento assinado eletronicamente por **Joao Paulo Da Silva Fonseca**, **Professor do Magistério Superior**, em 24/07/2024, às 11:50, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020 .

Documento assinado eletronicamente por **Marco Antonio Assfalk De Oliveira**, **Professor do Magistério Superior**, em 24/07/2024, às 17:38, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020 .

Documento assinado eletronicamente por **Weslley Da Silva Alves**, **Usuário Externo**, em 24/07/2024, às 18:19, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

A autenticidade deste documento pode ser conferida no site https://sei.ufg.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **4693351** e o código CRC **6304B46C**.

Referência: Processo nº 23070.017776/2024-16 SEI nº 4693351

Integração entre soluções inspiradas em planejamento automático e ambientes didáticos práticos baseados em CLP

Autor: David Benetti

***Resumo*— A Indústria 4.0 é um conceito caracterizado pela utilização de sistemas inteligentes, automatizados e interconectados com o intuito de aumentar a produtividade, a eficiência e a flexibilidade nos processos de produção. No entanto a aplicação de sistemas como o planejamento automático ainda é limitada. Enquanto os sistemas reais utilizam Controladores Lógico Programáveis (CLPs) com linguagem Ladder, os planejadores automáticos utilizam a linguagem Planning Domain Definition Language (PDDL). A discrepância entre essas tecnologias representa uma dificuldade em integrá-las. Este artigo explora a implementação de planejadores automáticos com CLPs e apresenta o desenvolvimento de um exemplo prático de integração de uma bancada didática com sistemas de planejamento automático utilizando uma solução de software desenvolvida em python.**

***Palavras-chave*—CLP, Indústria 4.0, Integração de sistemas, Planejador automático, Python.**

***Abstract*— Industry 4.0 is a concept characterized by the use of intelligent, automated and interconnected systems to increase productivity, efficiency and flexibility in production processes. However, the application of systems such as automatic planning is still limited. While real systems use Programmable Logic Controller (PLC) type equipment with Ladder language, automatic planners use Planning Domain Definition Language (PDDL). The discrepancy between these technologies represents a difficulty in integrating them. This article explores the implementation of automatic planners with PLCs and presents the development of a practical example of integrating a didactic testing bench with automated planning systems using a software solution developed in Python.**

***Index Terms*—AI Planning, Industry 4.0, Systems integration, PLC, Python.**

I. Introdução

A Indústria 4.0 está revolucionando o setor de manufatura por meio da implementação de sistemas inteligentes, automatizados e em rede que visam melhorar a flexibilidade, a eficiência e a produtividade dos processos de produção.

Apesar dos avanços significativos e dos possíveis benefícios da Indústria 4.0, a aplicação de sistemas de planejamento automático continua limitada por vários desafios. Os planejadores automáticos, que geram sequências de ações para atingir metas específicas, geralmente usam a linguagem *Planning Domain Definition Language* (PDDL). Essa linguagem é adequada para definir problemas complexos de planejamento, mas não é diretamente compatível com as linguagens operacionais usadas em sistemas industriais do mundo real, como a linguagem Ladder empregada pelos Controladores Lógico Programáveis (CLP). Os CLPs desempenham um papel importante na automação industrial, centralizando os dados dos sensores e controlando os atuadores para executar tarefas predefinidas.

Um grande desafio de integração é a diferença entre o controle prático e em tempo real oferecido pelos CLPs e os recursos de planejamento automático. Este artigo explora o uso de planejadores automáticos com CLPs e descreve uma solução de software baseada em Python, utilizando o protocolo Modbus TCP, para integrar uma bancada de testes didáticos com sistemas de planejamento automático em um exemplo do mundo real.

II. Fundamentação Teórica

A. *Indústria 4.0*

A indústria 4.0 é a quarta revolução industrial, que marca uma nova revolução na manufatura, na qual as operações industriais são inteligentes e conectadas. Essa transformação é apoiada por várias tecnologias que facilitam a integração e a harmonização de processos em todos os setores.

1) *Principais tecnologias*

A Indústria 4.0 baseia-se em várias tecnologias fundamentais que formam a base de seus desenvolvimentos. Por exemplo, a Internet das Coisas, que é a conexão de dispositivos e sensores para geração e compartilhamento de dados; a análise de Big Data, que lida com grandes quantidades de dados para serem transformados em informações úteis; a inteligência artificial e o aprendizado de máquina, que permitem que o sistema de computador aprenda sozinho a partir dos dados e tome decisões; e os sistemas ciber físicos, que combinam computação com componentes físicos para criar sistemas inteligentes [2]. Além disso, a robótica, a manufatura aditiva, também conhecida como impressão 3D, e a realidade aumentada expandem as oportunidades, melhorando a flexibilidade organizacional e a capacidade de produzir bens de maior valor em um período mais curto [3].

2) *Impacto na manufatura*

A Indústria 4.0 tem grande importância no campo da manufatura. Com o uso eficaz da Internet das Coisas e da análise de Big Data, os fabricantes podem monitorar e controlar os processos de produção, aumentando assim a eficiência e a produtividade. A inteligência artificial e o aprendizado de máquina também desempenham um papel crucial na técnica de manutenção preditiva que ajuda a identificar a probabilidade de falha do equipamento e a evitar

sua recorrência [4]. Além disso, o uso de sistemas ciber físicos e robótica ajuda a aprimorar a automação e a precisão na produção, o que leva a uma melhor qualidade dos produtos e a baixas chances de erro humano [5].

Ademais, a Indústria 4.0 permite maior personalização e a flexibilidade na produção. A manufatura aditiva permite a produção de peças complexas e personalizadas em um curto espaço de tempo, e a realidade aumentada auxilia os funcionários oferecendo-lhes instruções sobre como executar um determinado processo de forma eficaz e sem cometer erros. Essas melhorias não só aumentam a produtividade e a lucratividade dos estágios de fabricação subsequentes, mas também permitem uma melhor adaptação à dinâmica do mercado e às necessidades dos clientes, aumentando assim a competitividade e a inovação em toda a cadeia de produção [6].

B. Automação de processos

A automação de processos é o uso de várias técnicas que ajudam na automação da maioria das operações industriais, aumentando assim a eficiência, diminuindo o número de intervenções de pessoas e aumentando a precisão. Diferentes tipos de tecnologias de automação podem ser úteis nos setores, tornando-os mais eficientes em termos de operação, gasto de recursos e controle de qualidade dos produtos. O objetivo da automação de processos é obter uma capacidade de produção maior e reduzir ao máximo os custos e os erros humanos. Os sistemas de automação incluem sistemas de controle, como os sistemas de controle distribuído, os sistemas de controle de supervisão e aquisição de dados (SCADA) e os controladores lógico programáveis (CLPs) [7].

As possíveis vantagens da automação de processos são inúmeras, como maior produtividade, maior segurança, melhor qualidade do produto e custos mais baixos das operações. O controle automatizado implica que não é necessário nenhum administrador humano, que pode trabalhar ininterruptamente sem se desgastar ou se esgotar. Além disso, a automação reduz as possibilidades de erro humano, o que aumenta a segurança em condições de risco [8].

No entanto, a implementação da automação de processos também tem seus problemas. Algumas dessas tecnologias de automação podem ser caras no início, quando estão sendo implementadas, e podem exigir pessoal com treinamento especial para operá-las. Às vezes, a integração com outros processos na linha de produção de uma organização pode levar tempo e esforço. Além disso, com o aumento do nível dos sistemas de automação, eles também correm alto risco de invasão e outros desafios de segurança, portanto, exigem segurança adequada [9].

C. Controlador lógico programável (CLP)

Um controlador lógico programável (CLP) é um tipo especial de computador usado para gerenciar processos e equipamentos de fabricação. Eles foram projetados para serem usados em aplicações industriais abrasivas e permitem um gerenciamento preciso e em tempo real. Os CPLs são portáteis e podem ser programados eletronicamente para realizar diversas tarefas de controle, como se vê nos sistemas de automação atuais [10].

Os CLPs foram desenvolvidos no final da década de 1960 e, com o avanço da tecnologia, houve melhorias ao longo dos anos. Originalmente usados para superar os problemas do controle baseado em relés, os primeiros CLPs eram lentos, simples e capazes de executar apenas algumas funções. No mundo atual, há um aprimoramento contínuo da tecnologia que resultou na criação de CLPs mais aperfeiçoados, com maior capacidade de processamento, mais conexões e mais flexibilidade. Os CLPs modernos estão disponíveis em diferentes variedades e tamanhos: compactos para uso em pequena escala, modulares para uso em sistemas complexos e montados em rack para uso em grandes aplicações industriais.

Os CLPs são empregados em vários setores, principalmente no setor de produção, para automatizar procedimentos, operar equipamentos e gerenciar outros sistemas. Eles gerenciam linhas de montagem, robótica e produção em geral, e isso é feito de forma precisa e otimizada. Em serviços públicos, os CLPs são aplicados a seções de tratamento de água, fornecimento de energia elétrica e outros setores importantes, melhorando sua confiabilidade e segurança.

D. Linguagem Ladder

A linguagem Ladder, também conhecida como diagrama Ladder, é uma linguagem de programação usada para escrever software usado em controladores lógicos programáveis. A lógica Ladder é modelada com base nos diagramas lógicos de relé que foram adotados pelos sistemas de controle elétrico em seu desenvolvimento inicial. Os processos de controle são indicados graficamente e todos os diferentes componentes elétricos, como interruptores, relés e temporizadores, entre outros, são ilustrados simbolicamente [12]. Isso torna a lógica Ladder fácil de ser entendida por engenheiros e técnicos, pois ela se baseia na lógica tradicional de relés em operação.

A lógica Ladder é composta de Rungs em que cada um é uma regra de controle. Eles também devem ser lidos da esquerda para a direita e de cima para baixo, como no método convencional de leitura de um livro. Alguns dos componentes básicos da linguagem Ladder são mostrados abaixo:

1. Contatos: Entradas de controle que simulam condições de saída, interruptores ou sensores. Eles também são classificados como contatos normalmente abertos e normalmente fechados.
2. Bobinas: Prescrevem respostas de saída, como a comutação de um relé ou a energização de um motor. Isso significa que, quando a lógica do Rung é verdadeira, a bobina recebe energia ou é considerada energizada.

E. Planejamento automático

Os planejadores automáticos são ferramentas computacionais que produzem um plano que inclui as ações necessárias para atingir um determinado objetivo quando um conjunto de parâmetros e restrições é fornecida. Um problema de planejamento é um problema em que temos algum estado inicial e desejamos transformá-lo em um estado final através da aplicação de um conjunto de ações [13].

A linguagem *Planning Domain Definition Language* (PDDL) é uma família de linguagens que permite definir um problema de planejamento. Como o planejamento evoluiu, a linguagem usada para descrevê-lo também evoluiu e, portanto, há muitas versões da PDDL disponíveis com diferentes níveis de expressividade.

Dentre as versões PDDL a versão 1.2 formou a base da Competição AIPS de 1998 [14]. As ideias por trás da STRIPS [15], uma espécie de linguagem precursora que usava um padrão de design semelhante para definir problemas por meio do uso de predicados e ações, serviram como base principal para a versão PDDL 1.2. Dois componentes formam um problema na PDDL: um arquivo de problema e um domínio:

1. Domínio: Na PDDL 1.2, um arquivo de domínio descreve os recursos “universais” de um problema. Em essência, esses são os elementos que permanecem constantes, independentemente do problema específico que estamos tentando resolver. Isso se refere principalmente aos diferentes tipos de objetos, predicados e ações que podem ser incluídos no modelo no formato PDDL 1.2. Os tipos são usados para limitar os itens que podem ser incluídos nos parâmetros de uma ação. Podemos expressar ações e predicados gerais e especializados usando tipos e subtipos, por exemplo. Os predicados. Os predicados são declarações lógicas que carcterizam os atributos de um objeto ou conexões com outros objetos no domínio. Inicialmente os predicados são declarados falsos quando assumidos e podem assumir verdadeiros ou falsos a qualquer ponto no plano. As ações definem transformações na condição global, nas quais são divididas em três seções distintas. A primeira seção é a de parâmetros na qual especifica as coisas que estamos realizando a ação. A segunda é a seção de precondições que é composta por uma série de conjunções e disjunções de predicados que devem ser satisfeitas para que a ação seja aplicada. A terceira é a seção de efeitos que define quais valores devem ser definidos como verdadeiros ou falsos se uma ação for aplicada.

2. Problema: O arquivo problema forma a segunda metade de um problema de planejamento. O problema define precisamente quais objetos existem, o que é verdade sobre eles e, por fim, qual é o objetivo final, ou seja, qual estado do sistema é desejado quando o plano é finalizado.

Enquanto a linguagem PDDL permite descrição do domínio e do problema, os planejadores de inteligência artificial formam a segunda parte da solução de problemas de planejamento, permitindo ler e tentar resolver os problema e domínio PDDL. Existe uma grande variedade de planejadores disponíveis, sendo alguns superiores ou inferiores em geral ou em determinadas situações.

F. Python

Python é uma linguagem de programação moderna, de propósito geral e de alto nível, conhecida por sua clareza e facilidade de uso. Sua execução apresenta vários paradigmas de programação, como programação procedural, orientada a objetos e funcional, o que a torna versátil em sua aplicação. A biblioteca padrão do Python é elaborada e um número substancial de pacotes de terceiros pode ser muito útil para diferentes tarefas, desde o desenvolvimento da Web até a análise de dados.

Python possui várias bibliotecas que podem ser usadas para automação industrial. Essas bibliotecas podem ajudar em tarefas como o controle de hardware, a comunicação com protocolos industriais e a construção de sistemas de automação. Alguns exemplos de bibliotecas populares são:

1. PySerial: Essa biblioteca é usada para fazer interface com portas seriais e permitir que o Python interaja com vários instrumentos industriais, como sensores e atuadores.

2. Pandas: O Pandas é uma ferramenta importante para trabalhar com grandes conjuntos de dados durante a análise de dados e para a automação da manipulação de dados e dos relatórios.

3. NumPy: Oferece matrizes e matrizes multidimensionais grandes e uma ampla variedade de operações matemáticas a serem processadas, que são cruciais em cálculos numéricos na automação.

4. SciPy: O módulo SciPy amplia a funcionalidade do NumPy e contém funções otimizadas para cálculos científicos e técnicos.

5. Matplotlib: Essa biblioteca é usada para desenvolver gráficos estáticos, interativos e de animação que podem ser usados em dados em tempo real de processos industriais.

6. RPi.GPIO: uma biblioteca para programar os pinos GPIO do Raspberry Pi e a única maneira que permite que o Python gerencie atividades de hardware.

7. PyModbus: Uma biblioteca que lida com o protocolo Modbus (cliente e servidor) e que pode ser utilizada em ambientes industriais para se comunicar com PLCs, sensores e outros equipamentos.

8. PyScada: Um sistema SCADA de código aberto para Python. Pode ser utilizado para monitoramento e controle industrial.

G. Modbus TCP/IP

O protocolo Modbus TCP/IP é uma versão do Modbus que adota o protocolo TCP/IP para transmissão em redes Ethernet. Essa extensão permite que os dispositivos Modbus operem em grandes distâncias e façam interface com adaptadores de rede ou sistemas de LAN e Internet. O Modbus TCP/IP mantém o esquema mestre-escravo, mas transmite dados usando dispositivos de rede padrão, como roteadores e switches. A tecnologia TCP/IP oferece o recurso de detecção de erros e retransmissão de quaisquer dados perdidos.

III. METODOLOGIA

O ambiente prático didático escolhido para estudo de integração foi um sistema de separação de peças composto pela bancada Exsto XC243 e um CLP industrial Schneider Electric TM221CE16T. Este sistema representa um ambiente real de seleção de produtos em uma indústria.

Essa bancada é composta por uma esteira transportadora, painel de conexões, sensores, dispositivos elétricos e sistema pneumático. Este conjunto é montado em um rack de perfilado de alumínio e pés com rodas para facilitar o transporte.

A esteira transportadora é tracionada por um motor DC e ao longo de seu percurso possui diversos sensores para identificação das peças. Dentre os sensores presentes estão dois sensores óticos, um sensor capacitivo e um sensor indutivo que permitem a classificação de peças em relação ao tamanho e tipo de material. Os cilindros pneumáticos presentes são compostos por dois atuadores de ação simples com êmbolo magnético posicionados após os sensores de classificação e um terceiro atuador de ação dupla com êmbolo magnético posicionado mais além. Três caixas laterais para separação de peças são localizadas em frente a cada um dos atuadores e uma quarta e última caixa é localizada ao final do percurso da esteira, destinada para peças que não foram desviadas. Cada uma das caixas de descarte possui um sensor de final de curso que permite a detecção da chegada de peças.

Fig. 1. Bancada didática Exsto XC243

O próximo tópico apresenta a modelagem dessa bancada para um ambiente prático de automação.

A. Modelagem da bancada

Para ser possível aplicar uma solução de planejamento automático na bancada didática, é necessário realizar a modelagem desta como um domínio de planejamento. Inicialmente para realizar a modelagem da bancada em um domínio PDDL prático de automação avaliam-se os casos de uso. A bancada possui atuadores pneumáticos que podem ser estendidos ou retraídos, uma esteira que pode ser ativada e desativada e caixas de descarte com sensores de final de curso. Inspirado nas características dessa bancada foi desenvolvido um modelo de automação baseado em clientes e fornecedores.

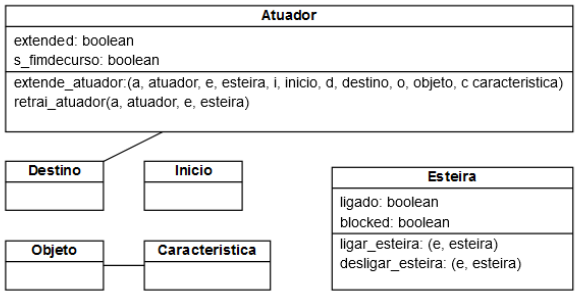

Fig. 2. Diagrama de classes da bancada

O problema desenvolvido consiste em um sistema de distribuição de peças entre clientes e fornecedores. Os fornecedores, representados pelos atuadores, entregam peças de diferentes tamanhos e características para os clientes, representados pelas caixas de descarte localizadas na lateral da esteira, de acordo com a demanda de cada cliente. Os sensores são responsáveis pela identificação da peça enquanto os atuadores são responsáveis pela entrega aos clientes.

B. Planning Dommains

O *website* http://planning.dommains/ agrega diversas ferramentas de planejamento automático utilizando o formato PDDL. Dentre as ferramentas disponíveis nesse *website* estão o editor online que permite editar arquivos PDDL e o componente de solução de problemas. Além do editor online, é possível enviar problemas e domínios em formato PDDL através de solicitações POST e obter planos resultados.

C. Programação em Ladder

Os CLPs quando acessados através do protocolo Modbus TCP não permitem acesso direto às entradas e saídas. Utilizando o software Ecostruxure Machine Expert – Basic foi desenvolvido um programa em linguagem Ladder que associa a identificação de peças de tamanhos e características diferentes através dos sensores e de ativação dos atuadores e esteiras a endereços de memória que podem ser lidos e alterados através do protocolo de comunicação Modbus TCP. A sequência de ações gerada pelo planejador possui um conjunto de pré-condições e pós-condições que determinam se um determinado estado foi atingido, e se a próxima ação pode ser executada. As ações geradas pelo planejador foram desenvolvidas para que estejam relacionadas ao ativamento de um conjunto de contatos lógicos de memórias auxiliares no CLP. Através da leitura dos valores de tais memórias, é possível identificar se a pré-condição e pôs-condição de cada ação do plano foi atingida. A figura 3 apresenta a tabela de atribuição de conexões dos elementos da bancada ao CLP.

DE	Identificação	Integrada
Entradas		
%I0.0	SO_M	"Sensor óptico-média"
%I0.1	SO_G	"Sensor óptico-grande"
%I0.2	S_Metálico	"Sensor indutivo (Metálico)"
%I0.3	SC	"Sensor capacitivo"
%I0.4	FC_2	"Fim de curso cliente 2 "
%I0.5	FC_1	"Fim de curso cliente 1 "
%I0.6	FC_3	"Fim de curso cliente 3 "
%I0.7	FC_4	"Fim de curso repositório "
%I0.8	Esteira	"Esteira do sistema"
Saídas		
%Q0.0	Motor_Esteira	"Ativa o motor da esteira"
%Q0.1	AT_1	"Ativa o atuador do cliente 1"
%Q0.2	AT_2	"Ativa o atuador do cliente 2"
%Q0.3	AT_3_Avanço	"Ativa o avanço do atuador do cliente 3"
%Q0.5	AT_3_Retorno	"Ativa o retorno do atuador do cliente 3"

Fig. 3. Tabela de atribuição do Sistema separador de peças

A figura 4 apresenta a tabela de relação entre os predicados das ações e os endereços de memória do CLP.

Identificação	Predicado Ação	Endereço de memória (CLP)	Saída/Entrada digital (CLP)
Entradas			
Liga Esteira	(ligado esteira)	%M0	Saída digital %Q0.0
Avança AP1	(extended atuador_simples1)	%M1	Saída digital %Q0.1
Avança AP2	(extended atuador_simples2)	%M2	Saída digital %Q0.2
Avança AP3	(extended atuador_duplo1)	%M3	Saída digital %Q0.3
Retrai AP3	-	%M4	Saída digital %Q0.5
Saídas			
Final de curso 1	(S_fimdecurso atuador_simples1)	%M14	Entrada digital %I0.5
Final de curso 2	(S_fimdecurso atuador_simples2)	%M15	Entrada digital %I0.4
Final de curso 3	(S_fimdecurso atuador_duplo1)	%M16	Entrada digital %I0.6
Final de curso 4	-	%M17	Entrada digital %I0.7
Pequena não metal ID	(type itemXpeqnmet1)	%M20	-
Pequena metal ID	(type itemXpeqmet1)	%M21	-
Média não metal ID	(type itemXmednmet1)	%M22	-
Média metal ID	(type itemXmedmet1)	%M23	-
Grande não metal ID	(type itemXgrdnmet1)	%M24	-
Grande metal ID	(type itemXgrdmet1)	%M25	-

Fig. 4. Tabela de associação de predicados e endereços de memória

D. Desenvolvimento de uma API em Python

Após realizada a modelagem da bancada para um problema de planejamento automático e desenvolvido um programa em linguagem Ladder, que associa as entradas e saídas do CLP da bancada de separação aos possíveis estados e ações do domínio de planejamento, foi desenvolvido uma aplicação em Python que permite integrar as soluções geradas pelo planejador automático com a bancada didática. No desenvolvimento da API foram utilizadas algumas bibliotecas Python.

Threading é uma biblioteca Python que oferece os meios para iniciar e interromper threads dentro de um programa. Threads são pequenos processos que compartilham memória, permitindo a execução simultânea de várias tarefas. As tarefas como leitura e gravação de arquivos, gerenciamento de interfaces de usuário e operações de rede, podem se beneficiar muito com isso. Mecanismos de sincronização são utilizados para garantir que as threads não entrem em conflito umas com as outras, especialmente quando tentam acessar recursos partilhados.

Requests é uma biblioteca usada para fazer solicitações HTTP. Ela oculta as dificuldades envolvidas no envio de solicitações HTTP por trás de uma API simples, permitindo que os desenvolvedores se comuniquem facilmente com serviços da *web* e APIs.

Pymodbus é uma biblioteca para comunicação com dispositivos que usam o protocolo Modbus. Oferece implementações para cliente e servidor, também conhecidas como mestre e escravo. Pode ser usado para criar um servidor Modbus que responde a solicitações de clientes ou um cliente que envia solicitações a dispositivos Modbus.

O programa desenvolvido contém os seguintes componentes:

I. Modbus.py: Contém a implementação de métodos, que utilizam a biblioteca Pymodbus, para realizar a leitura e escrita dos endereços de memória do CLP.

II. Main_program.py: Contém a lógica principal da API. Inicialmente é obtido a solução do problema através da API do website Planning Domains. Utilizando a biblioteca requests é realizado um método POST contendo os arquivos problema e domínio, localizados na pasta local. A resposta do método POST contém a solução, obtida através do planejador LAMA [18]. Para cada ação da solução é realizada a verificação dos valores registrados nos endereços de memória do CLP, utilizando o protocolo Modbus. Quando as pré-condições são atingidas a ação é executada. É realizada então a verificação dos efeitos da ação. Quando os efeitos são atingidos a próxima ação é analisada e caso não exista nenhuma ação restante o programa finaliza.

III. Program_GUI: Arquivo que contém a implementação de uma interface gráfica. Permite a definição do endereço IP do CLP e a atualização das demandas dos clientes. Possui uma representação gráfica da bancada.

IV. Requirements.txt: Contém uma lista de dependências do projeto.

A figura 5 apresenta o diagrama de funcionamento do sistema proposto.

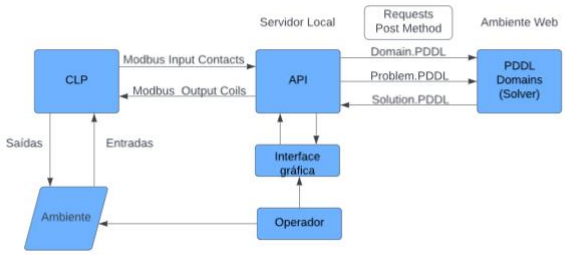

Fig. 5. Diagrama de funcionamento do programa

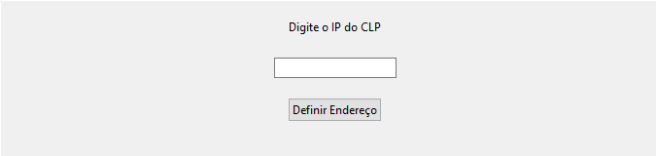

Fig. 6. Interface inicial do programa

O programa desenvolvido apresenta uma interface gráfica como representado pela figura 6. A tela inicial permite com

que o usuário defina o endereço IP do CLP. Após inserir o endereço no campo disponível e pressionar o botão definir endereço, é apresentado uma interface como representado na figura 7.

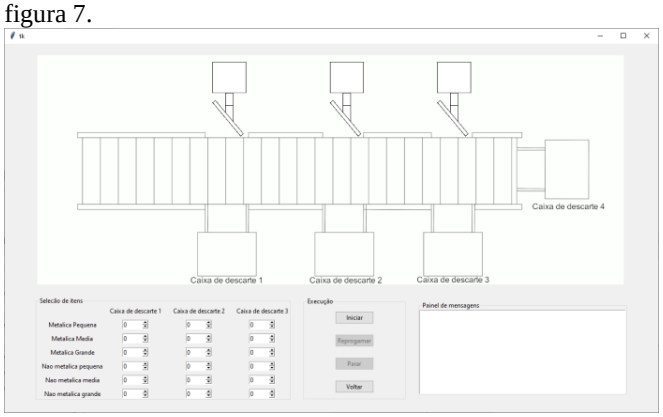

Fig. 7. Interface de execução do programa

O painel representado na figura 7 apresenta uma representação gráfica do problema, um menu de seleção de itens que permite a definição da demanda para cada caixa de descarte, um menu de execução que permite o controle da execução do programa e um painel de mensagens que exibe informações sobre a execução do programa.

A figura 8 apresenta um diagrama de sequência para um caso de execução básico do programa.

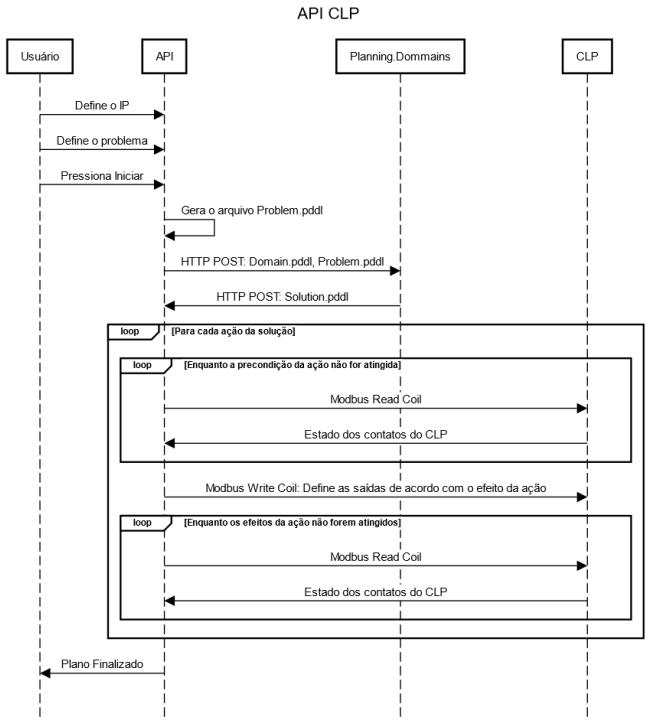

Fig. 8. Diagrama de sequência

E. Exemplos de execução da ferramenta proposta

A seguir é exemplificado a execução da API proposta para diferentes cenários.

1) Cenário básico de execução

Este cenário representa um caso básico de utilização. Quando inicializado, o programa apresenta a tela inicial de definição do endereço do controlador. O Usuário deve inserir o endereço IP do controlador e pressionar o botão definir endereço.

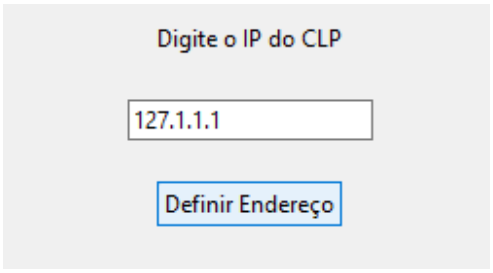

Fig. 9. Definindo o endereço do controlador

Após o endereço IP ser definido o usuário deve selecionar as demandas de itens como exibido na figura 10.

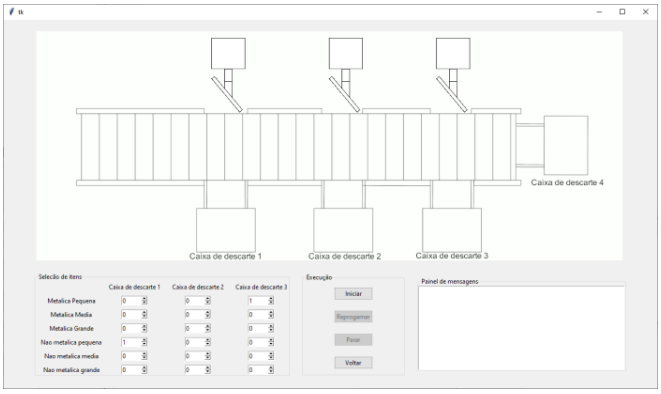

Fig. 10. Definindo as demandas

Neste caso de exemplo foi selecionada a demanda de uma peça pequena não metálica para a caixa de descarte 1 e uma peça metálica pequena para a caixa de descarte 3. Após selecionadas as demandas de cada caixa, o botão iniciar é selecionado e a execução se inicia.

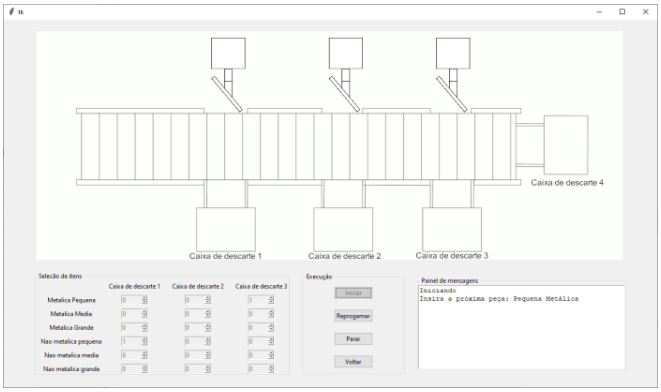

Fig. 11. Iniciando a execução

O painel de mensagens exibe a informação sobre a próxima peça a ser inserida como exibido na figura 11. O operador deve então inserir a peça solicitada, que neste caso uma peça pequena metálica.

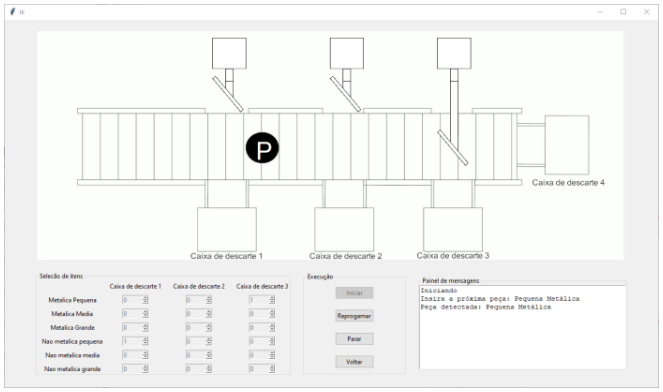

Fig. 12. Ilustração de uma peça pequena metálica detectada

A figura 12 demonstra a peça pequena metálica sendo movimentada na esteira e o atuador 3 posicionado para desviá-la à respectiva caixa de descarte.

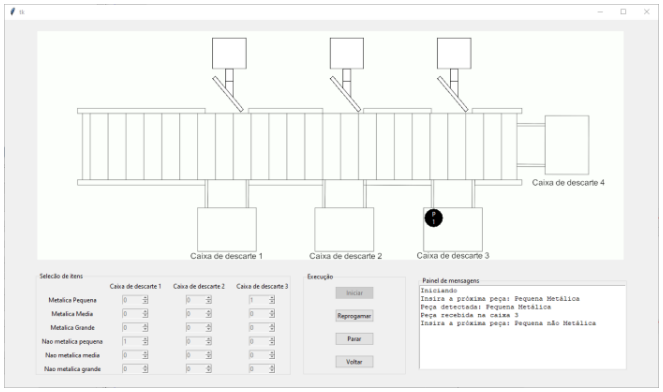

Fig. 13. Ilustração de uma peça pequena metálica separada para a caixa de descarte 3

A figura 13 apresenta a peça inserida armazenada na caixa de descarte 3 após ser separada. O painel de mensagens exibe a informação sobre a próxima peça que deve ser inserida. Neste caso a próxima peça é uma peça pequena não metálica.

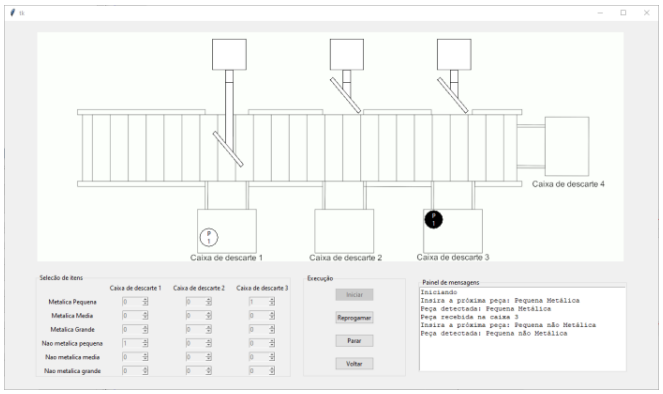

Fig. 14. Ilustração do estado intermediário em que a segunda peça foi separada na caixa de descarte 1

A figura 14 apresenta a peça pequena não metálica separada na caixa de descarte 1.

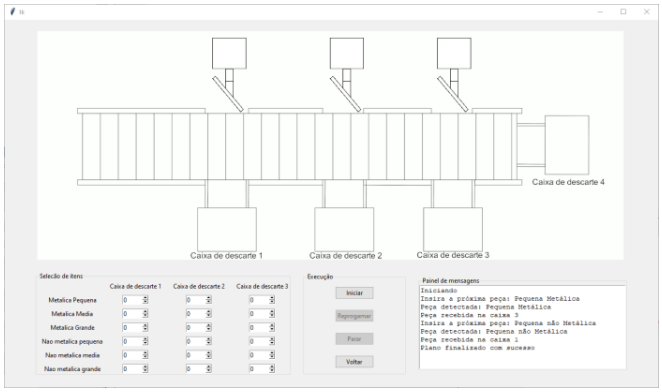

Fig. 15. Plano Finalizado

A figura 15 apresenta o programa finalizado com sucesso após a peça pequena não metálica ser recebida na caixa de descarte 1.

2) Cenário com reprogramação

Este cenário apresenta um exemplo de uso da funcionalidade reprogramar. A reprogramação permite que a execução do plano seja pausada e as demandas de cada caixa de descarte sejam atualizadas.

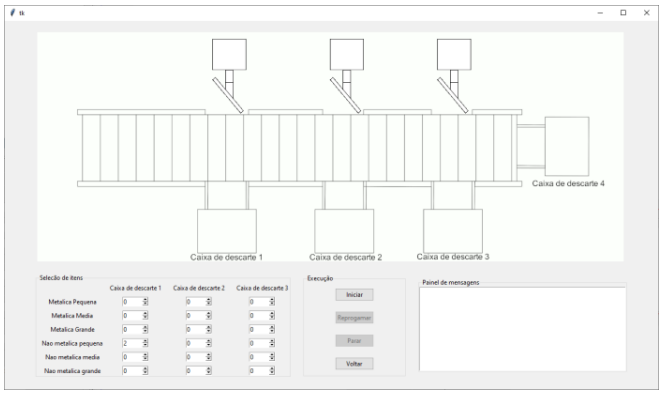

Fig. 16. Exemplo de uma execução com reprogramação

Neste cenário foi definida inicialmente a demanda de duas peças pequenas não metálicas para a caixa de descarte 1.

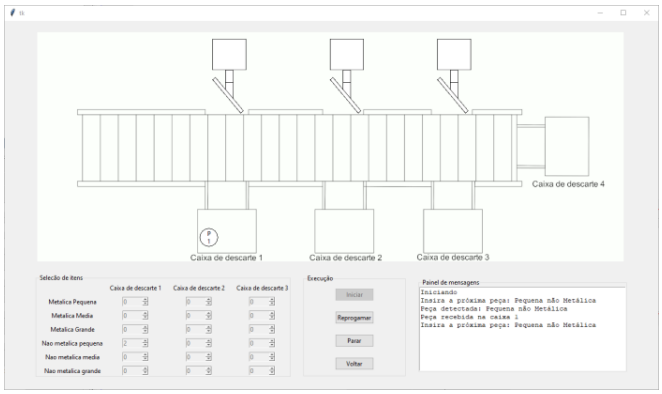

Fig. 17. Peça inicial descartada

A figura 17 apresenta a primeira peça descartada na caixa 1.

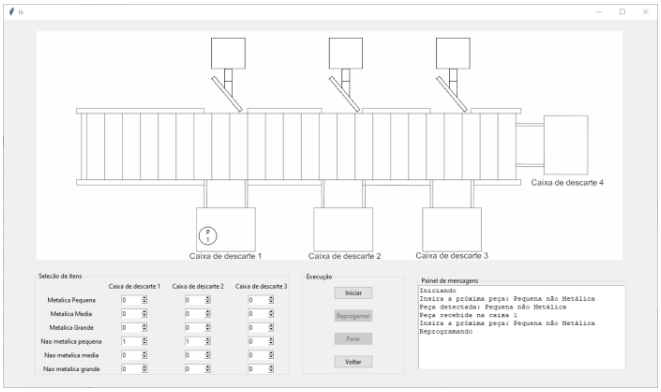

Fig. 18. Detalhe da atualização de demandas após o botão reprogramar ter sido pressionado

A figura 18 exibe as demandas atualizadas após o botão reprogramar ter sido selecionado.

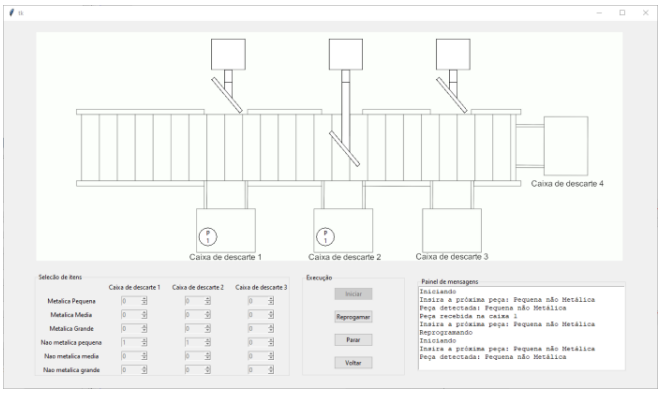

Fig. 19. Reinício após reprogramação

Após as atualizar as demandas a execução é resumida selecionando o botão iniciar. A Figura 19 exibe a peças separadas após a reprogramação.

3) Cenário com peça inserida incorretamente

Este cenário apresenta um exemplo de uma peça inserida incorretamente.

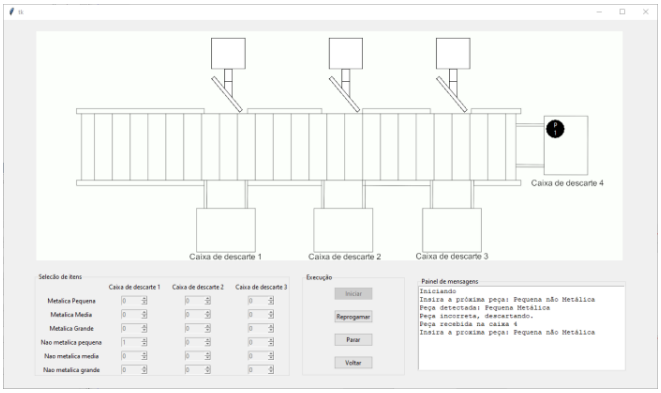

Fig. 20. Peça incorreta inserida

A figura 20 apresenta uma peça incorreta descartada na caixa 4.

IV. RESULTADOS E CONCLUSÕES

Através do uso da linguagem de programação Python e suas bibliotecas, foi desenvolvido uma aplicação capaz de integrar as soluções de um planejador automático com a bancada didática. Foi realizado a modelagem de uma bancada didática de planejamento automático e plano foi transmitido para o CLP TM221CE16T da Schneider Electric.

REFERÊNCIAS

[1] Saurabh Vaidya, Prashant Ambad, Santosh Bhosle, Industry 4.0 – A Glimpse, Procedia Manufacturing, Volume 20, 2018, Pages 233-238, ISSN 2351-9789, https://doi.org/10.1016/j.promfg.2018.02.034

[2] K. Zhou, T. Liu, and L. Zhou, "Industry 4.0: Towards future industrial opportunities and challenges," in 2015 12th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD), 2015, pp. 2147-2152.

[3] V. Alcácer and V. Cruz-Machado, "Scanning the Industry 4.0: A Literature Review on Technologies for Manufacturing Systems," Engineering Science and Technology, an International Journal, vol. 22, no. 3, pp. 899–919, Jun. 2019, doi: https://doi.org/10.1016/j.jestch.2019.01.006

[4] J. Lee, B. Bagheri, and H.-A. Kao, "A Cyber-Physical Systems architecture for Industry 4.0-based manufacturing systems," Manufacturing Letters, vol. 3, pp. 18-23, 2015.

[5] R. Drath and A. Horch, "Industrie 4.0: Hit or hype?" IEEE Industrial Electronics Magazine, vol. 8, no. 2, pp. 56-58, 2014.

[6] G. Orzes, E. Rauch, S. Bednar, and R. Poklemba, "Industry 4.0 Implementation Barriers in Small and Medium Sized Enterprises: A Focus Group Study," IEEE Xplore, Dec. 01, 2018. https://ieeexplore.ieee.org/document/8607477

[7] J. Love, "Process Automation Handbook: A Guide to Theory and Practice," Springer, 2005.

[8] V. V. Patel, M. V. Liarokapis and A. M. Dollar, "Open Robot Hardware: Progress, Benefits, Challenges, and Best Practices," in IEEE Robotics & Automation Magazine, vol. 30, no. 3, pp. 123-148, Sept. 2023, doi: 10.1109/MRA.2022.3225725.

[9] D. Bhamare, M. Zolanvari, A. Erbad, R. Jain, K. Khan, and N. Meskin, "Cybersecurity for industrial control systems: A survey," Computers & Security, vol. 89, p. 101677, Feb. 2020, doi: https://doi.org/10.1016/j.cose.2019.101677

[10] J. Stenerson, "Fundamentals of Programmable Logic Controllers, Sensors, and Communications," Pearson, 2013.

[11] M. M. Lashin, "Different Applications of Programmable Logic Controller (PLC)," International Journal of Computer Science, Engineering and Information Technology, vol. 4, no. 1, pp. 27–32, Feb. 2014, doi: https://doi.org/10.5121/ijcseit.2014.4103

[12] W. Bolton, "Programmable Logic Controllers," 5th ed., Newnes, 2009.

[13] M. Ghallab, D. Nau, and P. Traverso, "Automated Planning: Theory and Practice," Elsevier, 2004.

[14] Ghallab, M., Knoblock, C., Wilkins, D., Barrett, A., Christianson, D., Friedman, M., Kwok, C., Golden, K., Penberthy, S., Smith, D., Sun, Y., & Weld, D. (1998). PDDL - The Planning Domain Definition Language.

[15] Fikes, R. E., & Nilsson, N. J. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. Artificial Intelligence, 2(3–4), 189–208.

[16] Modbus Organization. (2006). Modbus Messaging on TCP/IP Implementation Guide V1.0b. Disponível em: https://modbus.org/docs/Modbus_Messaging_Implementation_Guide_V1_0b.pdf

[17] Modbus Organization. (2012). MODBUS APPLICATION PROTOCOL SPECIFICATION V1.1b3. Disponível em https://modbus.org/docs/Modbus_Messaging_Implementation_Guide_V1_0b.pdf

[18] Richter, S., Matthias Westphal. (2010). The LAMA Planner: Guiding Cost-Based Anytime Planning with Landmarks

[19] Schneider Electric. *EIO0000003287 - EcoStruxure Machine Expert - Basic, Guia de instruções.* Disponível em https://www.se.com/br/pt/download/document/EIO0000003287/

APÊNDICES

APÊNDICE A. Códigos da bancada de planejamento

```
(define (domain sorting)
    (:requirements :typing)

    (:types
        ;; tipos de objetos
        atuador
        objeto
        inicio destino - local
        esteira
        caracteristica
    )

    (:predicates
        ;;Estado do atuador
        (extended ?atuador - atuador)

        ;;Estado da esteira
        (ligado ?e - esteira)

        ;;Local do objeto
        (at ?o - objeto ?l - local)

        ;;Sensor de final de curso
        (s_findecurso ?a - atuador)

        ;;Tipo do objeto
        (type ?o - objeto ?c - caracteristica)

        (blocked ?e - esteira)

        (link ?a - atuador ?d - destino)

    )

    ;; ligar a esteira
    (:action ligar_esteira
        :parameters (?e - esteira)
        :precondition (and
            (not (ligado ?e))
        )
        :effect (and
            (ligado ?e)
        )
    )

    ;; desligar a esteira
    (:action desligar_esteira
        :parameters (?e - esteira)
        :precondition (and
            (not(blocked ?e))
            (ligado ?e)
        )
        :effect (and
            (not (ligado ?e))
        )
    )

    ;; Extende o atuador
    (:action extende_atuador
        :parameters(?e - esteira ?a - atuador ?i - inicio ?d - destino ?o - objeto ?c - caracteristica)
        :precondition (and
            (type ?o ?c)
            (ligado ?e)
            (not (extended ?a))
            (at ?o ?i)
            (not(blocked ?e))
            (link ?a ?d)
        )
        :effect (and
            (at ?o ?d)
            (not (at ?o ?i))
            (extended ?a)
            (s_findecurso ?a)
            (blocked ?e)
        )
    )

    ;; Retrai o atuador
    (:action retrai_atuador
        :parameters(?e - esteira ?a - atuador )
        :precondition (and
            (s_findecurso ?a)
            (extended ?a)
            (blocked ?e)
        )
        :effect (and
            (not (extended ?a))
            (not (s_findecurso ?a))
            (not(blocked ?e))
        )
    )
)
```

Fonte: Elaboração Própria.

```
(define (problem more)
(:domain sorting)

    (:objects
        atuador_simples1 atuador_simples2 atuador_duplo1 - atuador
        box1 box2 box3 - destino

        esteira - esteira
        inicio - inicio
        peqnmet1 peqmet1 mednmet1 medmet1 grdnmet1 grdmet1 - caracteristica

        item0 - objeto

    )

    (:init
    (link atuador_simples1 box1)
        (link atuador_simples2 box2)
        (link atuador_duplo1 box3)

        (type item0 grdmet1)

        (at item0 inicio)

        (not (ligado esteira))
    )

    (:goal (and
        (at item0 box1)

        (not (ligado esteira))
        (not (extended atuador_simples1))
        (not (extended atuador_simples2))
        (not (extended atuador_duplo1))
        )
    )
)
```

Fonte: Elaboração Própria.

```
(:action ligar_esteira
  :parameters (esteira)
  :precondition
    (and
      (not
        (ligado esteira)
      )
    )
  :effect
    (and
      (ligado esteira)
    )
)
(:action extende_atuador
  :parameters (esteira atuador_simples1 inicio box1 item0 peqnmet1)
  :precondition
    (and
      (type item0 peqnmet1)
      (ligado esteira)
      (not
        (extended atuador_simples1)
      )
      (at item0 inicio)
      (not
        (blocked esteira)
      )
      (link atuador_simples1 box1)
    )
  :effect
    (and
      (at item0 box1)
      (not
        (at item0 inicio)
      )
      (extended atuador_simples1)
      (s_fimdecurso atuador_simples1)
      (blocked esteira)
    )
)
(:action retrai_atuador
  :parameters (esteira atuador_simples1)
  :precondition
    (and
      (s_fimdecurso atuador_simples1)
      (extended atuador_simples1)
      (blocked esteira)
    )
  :effect
    (and
      (not
        (extended atuador_simples1)
      )
      (not
        (s_fimdecurso atuador_simples1)
      )
      (not
        (blocked esteira)
      )
    )
)
(:action desligar_esteira
  :parameters (esteira)
  :precondition
    (and
      (not
        (blocked esteira)
      )
      (ligado esteira)
    )
  :effect
    (and
      (not
        (ligado esteira)
      )
    )
)
```

Fonte: Elaboração Própria.

APÊNDICE B. Exemplo de códigos da API

```
class Mainloop():

    def __init__(self, ip):

        #Endereço Solver.Planning.Domains
        self.enderco_solver = 'https://solver.planning.domains:5001'
        self.package = '/package/lama-first/solve'

        #Instancia o cliente Modbus.
        self.client = modbus.ModbusTcpClient(ip)


        #Dicionario contendo o estado das entradas e saidas digitais do CLP.
        self.table = {
            'liga_esteira': False,
            'anvanca_ap1': False,
            'anvanca_ap2': False,
            'anvanca_ap3': False,
            'retrai_ap3': False,
            'fc_1': False,
            'fc_2': False,
            'fc_3': False,
            'fc_4': False,
            'peca_peqnmet': False,
            'peca_peqmet': False,
            'peca_mednmet': False,
            'peca_medmet': False,
            'peca_grandnmet': False,
            'peca_grandmet': False
        }

        # Enderco das entradas e saidas
        self.coil_addr = {
            'liga_esteira': 0,
            'anvanca_ap1': 1,
            'anvanca_ap2': 2,
            'anvanca_ap3': 3,
            'retrai_ap3': 4,
            'fc_1': 14,
            'fc_2': 15,
            'fc_3': 16,
            'fc_4': 17,
            'peca_peqnmet': 20,
            'peca_peqmet': 21,
            'peca_mednmet': 22,
            'peca_medmet': 23,
            'peca_grdnmet': 24,
            'peca_grdmet': 25
        }

        #Flags de status de funcionamento
        self.running = False
        self.stop = False
        self.state = True
        self.finalizado = False

        self.precondition_dict = {}
        self.effect_dict = {}
```

Fonte: Elaboração Própria.

```
def generate_problem(self):
    # Armazena as informacoes sobre o problema desejado
    items = {
        'cx1_peq_metal': int(self.spinbox2.get()) - self.caixa_descarte1['peca_peqmet'],
        'cx1_med_metal': int(self.spinbox3.get()) - self.caixa_descarte1['peca_medmet'],
        'cx1_grd_metal': int(self.spinbox4.get()) - self.caixa_descarte1['peca_grdmet'],
        'cx1_peq': int(self.spinbox5.get()) - self.caixa_descarte1['peca_peqnmet'],
        'cx1_med': int(self.spinbox6.get()) - self.caixa_descarte1['peca_mednmet'],
        'cx1_grd': int(self.spinbox7.get()) - self.caixa_descarte1['peca_grdnmet'],

        'cx2_peq_metal': int(self.spinbox8.get()) - self.caixa_descarte2['peca_peqmet'],
        'cx2_med_metal': int(self.spinbox9.get()) - self.caixa_descarte2['peca_medmet'],
        'cx2_grd_metal': int(self.spinbox10.get()) - self.caixa_descarte2['peca_grdmet'],
        'cx2_peq': int(self.spinbox11.get()) - self.caixa_descarte2['peca_peqnmet'],
        'cx2_med': int(self.spinbox12.get()) - self.caixa_descarte2['peca_mednmet'],
        'cx2_grd': int(self.spinbox13.get()) - self.caixa_descarte2['peca_grdnmet'],

        'cx3_peq_metal': int(self.spinbox14.get()) - self.caixa_descarte3['peca_peqmet'],
        'cx3_med_metal': int(self.spinbox15.get()) - self.caixa_descarte3['peca_medmet'],
        'cx3_grd_metal': int(self.spinbox16.get()) - self.caixa_descarte3['peca_grdmet'],
        'cx3_peq': int(self.spinbox17.get()) - self.caixa_descarte3['peca_peqnmet'],
        'cx3_med': int(self.spinbox18.get()) - self.caixa_descarte3['peca_mednmet'],
        'cx3_grd': int(self.spinbox19.get()) - self.caixa_descarte3['peca_grdnmet']
    }
    for key in items:
        if items[key] < 0:
            items[key] = 0

    # Gerando o problema a partir das informacoes inseridas
    lines = []

    n_box1 = items['cx1_peq_metal'] + items['cx1_med_metal'] + items['cx1_grd_metal'] + items['cx1_peq'] + items[
        'cx1_med'] + items['cx1_grd']
    n_box2 = items['cx2_peq_metal'] + items['cx2_med_metal'] + items['cx2_grd_metal'] + items['cx2_peq'] + items[
        'cx2_med'] + items['cx2_grd']
    n_box3 = items['cx3_peq_metal'] + items['cx3_med_metal'] + items['cx3_grd_metal'] + items['cx3_peq'] + items[
        'cx3_med'] + items['cx3_grd']

    n_peq_metal = items['cx1_peq_metal'] + items['cx2_peq_metal'] + items['cx3_peq_metal']
    n_med_metal = items['cx1_med_metal'] + items['cx2_med_metal'] + items['cx3_med_metal']
    n_grd_metal = items['cx1_grd_metal'] + items['cx2_grd_metal'] + items['cx3_grd_metal']

    n_peq = items['cx1_peq'] + items['cx2_peq'] + items['cx3_peq']
    n_med = items['cx1_med'] + items['cx2_med'] + items['cx3_med']
    n_grd = items['cx1_grd'] + items['cx2_grd'] + items['cx3_grd']

    n_items = sum(items.values())
```

Fonte: Elaboração Própria.

```
#Atualizando posição da peça em movimento
if self.moving_piece_data['moving']:

    if self.moving_piece_data['start'] is True:
        self.moving_piece_data['start'] = False

        Starting_piece_coord = [100, 200]
        match self.moving_piece_data['type']:
            case 'peca_peqnmet':
                self.canvas1.itemconfigure(self.peqnmet_moving_image, state=tk.NORMAL)
                self.canvas1.coords(self.peqnmet_moving_image, *Starting_piece_coord)
            case 'peca_peqmet':
                self.canvas1.itemconfigure(self.peqmet_moving_image, state=tk.NORMAL)
                self.canvas1.coords(self.peqmet_moving_image, *Starting_piece_coord)
            case 'peca_medmet':
                self.canvas1.itemconfigure(self.medmet_moving_image, state=tk.NORMAL)
                self.canvas1.coords(self.medmet_moving_image, *Starting_piece_coord)
            case 'peca_mednmet':
                self.canvas1.itemconfigure(self.mednmet_moving_image, state=tk.NORMAL)
                self.canvas1.coords(self.mednmet_moving_image, *Starting_piece_coord)
            case 'peca_grdnmet':
                self.canvas1.itemconfigure(self.grdnmet_moving_image, state=tk.NORMAL)
                self.canvas1.coords(self.grdnmet_moving_image, *Starting_piece_coord)
            case 'peca_grdmet':
                self.canvas1.itemconfigure(self.grdmet_moving_image, state=tk.NORMAL)
                self.canvas1.coords(self.grdmet_moving_image, *Starting_piece_coord)


    self.update_img_pos(self.moving_piece_data['type'])
```

Fonte: Elaboração Própria.

```python
class aUI:
    def __init__(self, master=None):
        self.ip = None
        self.thread_client = None
        self.client_instance = None
        self.end_que = Queue()
        self.pecas_que = Queue()
        self.descartando_pecaincorreta = False
        self.proximapeca = None

        #Armazena informações sobre a existencia de uma peça em movimento
        self.moving_piece_data = {
            'type': None,
            'destination': None,
            'moving': False,
            'x_coord': 0,
            'y_coor': 0,
            'start': False
        }
        self.item_dx = 10
        self.item_dy = 10

        #Armazena a quantidade de peças em cada caixa
        self.caixa_descarte1 = {
            'peca_pegnmet': 0,
            'peca_pegmet': 0,
            'peca_mednmet': 0,
            'peca_medmet': 0,
            'peca_grdnmet': 0,
            'peca_grdmet': 0
        }
        self.caixa_descarte2 = {
            'peca_pegnmet': 0,
            'peca_pegmet': 0,
            'peca_mednmet': 0,
            'peca_medmet': 0,
            'peca_grdnmet': 0,
            'peca_grdmet': 0
        }
        self.caixa_descarte3 = {
            'peca_pegnmet': 0,
            'peca_pegmet': 0,
            'peca_mednmet': 0,
            'peca_medmet': 0,
            'peca_grdnmet': 0,
            'peca_grdmet': 0
        }
        self.caixa_descarte4 = {
            'peca_pegnmet': 0,
            'peca_pegmet': 0,
            'peca_mednmet': 0,
            'peca_medmet': 0,
            'peca_grdnmet': 0,
            'peca_grdmet': 0
        }

        self.precondition_dict = {}
        self.table = {
            'liga_esteira': False,
            'avanca_ap1': False,
            'avanca_ap2': False,
            'avanca_ap3': False,
            'retrai_ap3': False,
            'fc_1': False,
            'fc_2': False,
            'fc_3': False,
            'fc_4': False,
            'peca_pegnmet': False,
            'peca_pegmet': False,
            'peca_mednmet': False,
            'peca_medmet': False,
            'peca_grandnmet': False,
            'peca_grandmet': False
        }
```

Fonte: Elaboração Própria.

```
def generate_problem(self):
    #Armazena as informacoes sobre o problema desejado
    items = {
        'cx2_peq_metal': self.spinbox1.get(),
        'cx2_med_metal': self.spinbox2.get(),
        'cx2_grd_metal': self.spinbox3.get(),
        'cx2_peq': self.spinbox4.get(),
        'cx2_med': self.spinbox5.get(),
        'cx2_grd': self.spinbox6.get(),

        'cx1_peq_metal': self.spinbox7.get(),
        'cx1_med_metal': self.spinbox8.get(),
        'cx1_grd_metal': self.spinbox9.get(),
        'cx1_peq': self.spinbox10.get(),
        'cx1_med': self.spinbox11.get(),
        'cx1_grd': self.spinbox12.get(),

        'cx3_peq_metal': self.spinbox13.get(),
        'cx3_med_metal': self.spinbox14.get(),
        'cx3_grd_metal': self.spinbox15.get(),
        'cx3_peq': self.spinbox16.get(),
        'cx3_med': self.spinbox17.get(),
        'cx3_grd': self.spinbox18.get()
    }

    #Gera a descricao do problema
    functions.generate_problemFile(items)
```

Fonte: Elaboração Própria.