GUILHERME VITOR DOS SANTOS RODRIGUES

IMPLEMENTAÇÃO DE PRD INTEGRADO A CONTROLE ADAPTATIVO APLICADA A ROBÔ LINEAR

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE ENGENHARIA MECÂNICA
2024

GUILHERME VITOR DOS SANTOS RODRIGUES

IMPLEMENTAÇÃO DE PRD INTEGRADO A CONTROLE ADAPTATIVO APLICADA A ROBÔ LINEAR

Trabalho apresentado ao Curso de Graduação em Engenharia Mecatrônica da Universidade Federal de Uberlândia, com parte dos requisitos para a obtenção do título de BACHAREL EM ENGENHARIA MECATRÔNICA.

Orientador: Prof. Dr. José Jean-Paul Zanlucchi de Souza Tavares

Uberlândia – MG

2024

Agradecimentos

À minha mãe Aurélia Cristina dos Santos, ao meu pai Antônio Luiz Rodrigues e meu irmão Vinicius André dos Santos Rodrigues por todo apoio e compreensão durante o curso e por sempre acreditarem em mim e me incentivarem.

A todos os meus amigos e familiares que direta ou indiretamente contribuíram para minha formação, com conselhos, palavras de conforto e apoio incondicional.

Ao Prof. Dr. José Jean-Paul Zanlucchi de Souza Tavares pela orientação e oportunidade de poder trabalhar em um projeto que me agregou bastante aprendizado e descobertas.

E, finalmente, à Universidade Federal de Uberlândia e à Faculdade de Engenharia Mecânica pela oportunidade de participar do curso de Engenharia Mecatrônica e por toda a infraestrutura oferecida durante estes anos como aluno.

Resumo

Os trabalhos anteriores utilizando redes de Petri com RFID, denominado *Petri Net inside RFID database* ou PNRD e a PNRD invertida(iPNRD) possibilitam a identificação de um produto e também permitem realizar uma atualização do estado do item com relação ao processo esperado em tempo real. Estes sistemas são limitados em relação ao tamanho do modelo do processo que explode exponencialmente com o número de itens que podem ser identificados além de não ter internamente um sistema de controle adaptativo que possa reconfigurar os movimentos dos robôs utilizados em possíveis exceções.A alternativa para este problema é a implementação de uma integração de um sistema de planejamento utilizando a PRD(*Predicate Inside RFID Data structure*). Para prova de conceito se pretende implementar este sistema em um robô cartesiano. Neste trabalho é realizado a confecção de um sistema completo PRD. Utilizando uma garra robótica e um sensor RFID acoplado à garra, sendo o sensor lido por um Arduino MEGA integrado com a Shield Ramps 1.4 diretamente conectados ao computador principal em que o robô cartesiano atua ao executar o sistema ativam módulos de um sistema discreto que são utilizados para um funcionamento completo.Como resultado, se obtém um sistema computacional capaz de lidar com diversos tipos de exceções, desde àquelas relativas a identificação do estado inicial e final aqui identificadas como estáticas. Como também das exceções que surgem ao longo da movimentação dos itens, nesse caso chamadas de dinâmicas.

Palavras-chave: robô cartesiano, PRD, RFID, Planejamento Automático.

Abstract

Previous works using Petri nets with RFID, called *Petri Net inside RFID database* or PNRD and the inverted PNRD (iPNRD), enable product identification and also allow updating the item's state with respect to the expected process in real time. These systems are limited in terms of the size of the process model, which exponentially explodes with the number of items that can be identified, and they do not internally have an adaptive control system that can reconfigure the movements of the robots used in possible exceptions. The alternative to this problem is the implementation of integration of a planning system using the PRD (*Predicate Inside RFID Data structure*). For proof of concept, it is intended to implement this system in a Cartesian robot. In this work, a complete PRD system is fabricated. Using a robotic gripper and an RFID sensor attached to the gripper, with the sensor read by an Arduino MEGA integrated with the Shield Ramps 1.4 directly connected to the main computer in which the Cartesian robot operates when executing the system, discrete system modules are activated, which are used for full operation. As a result, a computational system capable of dealing with various types of exceptions is obtained, from those related to the identification of the initial and final states identified here as static, to those that arise during the movement of items, in this case called dynamic.

Keywords: Cartesian robot, PRD, RFID, Automatic Planning.

Lista de Figuras

Glossário

RFID – Identificação por radiofrequência (*Radio-Frequency IDentification*)

RAMPS – Escudo RepRap Arduino Mega Pololu (*RepRap Arduino Mega Pololu Shield*)

PRD - Predicados Inseridos em Base de dados RFID(*Predicate inside RFID Data Structure*)

XML - Linguagem de Marcação de Hipertexto Extensível(*eXtensible Hypertext Markup Language*)

JSON - Notação de Objetos Javascript(*Javascript Object Notation*)

PDDL- Linguagem de Definição de domínio de Planejamento(*Planning Domain Definition Language*)

RegEx- Expressões Regulares(*Regular Expressions*)

ROS- Sistema Operativo Robótico(*Robot Operating System*)

USB- Porta Serial Universal(*Universal Serial Bus*)

IDE- Ambiente de Desenvolvimento Integrado(*Integrated Development Environment*)

GUI - Interface de Usuário Gráfica(*Graphic User Interface*)

PSS - Espaço de Estados Físico(*Physical State Space*)

UART - Transmissor/Receptor Assíncrono Universal(*Universal Asynchronous Receiver / Transmitter*)

I2C -Circuito Inter-Integrado(*Inter-Integrated Circuit*)

SPI - Interface Periférica Serial(*Serial Peripheral Interface*)

UML - Linguagem de Modelo Unificada(*Unified Modeling Language*)

Conteúdo

1 INTRODUÇÃO

Sistemas distribuídos computadorizados estão sendo cada vez mais utilizados com a recente demanda de soluções sem contato humano direto,como é o caso da obtenção automática de informação em relação aos produtos e serviços, e com a integração da conectividade entre dispositivos relacionados a cada atividade, aliado ao uso de aplicativos de tomada de decisão automática. Mas como é possível manter estável um sistema mesmo com intermitência de comunicação ethernet? As soluções devem ser cada vez mais autônomas para resolver por si mesmas o maior número de exceções ao processo.

Com o aumento do poder computacional atual fica possível utilizar aplicativos de inteligências artificiais, como é o exemplo o *Machine Learning* (ML), que permite que uma máquina perceber o mundo ou o ambiente de trabalho do agente como um ser humano faz, aprendendo e melhorando as experiências ganhou com a circunstância [**17**]. Esse é um assunto que tem destaque nas publicações, tanto referentes a robôs fixos [**12**] quanto robôs móveis [**2**]. Porém, ML se baseia fortemente em reconhecimento de imagem, que requer tanto um hardware de alto custo, como também alta demanda energética para processar os dados e imagens em tempo real. Além disso, todo alarde em relação à inteligência artificial foca no desenvolvimento de algoritmos em servidores sem restrição de hardware e altamente dependentes de comunicação, pouco considerando sistemas digitais ou embarcados aplicados diretamente em circuitos eletrônicos nos robôs físicos apoiados com a Internet das coisas(IoT) com acesso intermitente à computação em nuvem; o que dificulta sua aplicação em qualquer situação.

De acordo com [**19**] atualmente há dois métodos principais de programação de robôs. No modo *on-line* (chamado de aprendizagem modo) o programador configura manualmente a ponta do manipulador em determinados pontos (posições) e salva suas coordenadas. Em método *off-line*, o programa deve ser inserido usando um computador e depois carregado no controlador do robô. Em ambos os métodos é necessário especificar muitos detalhes em relação às formas dos caminhos de movimento e às posições que o robô deve alcançar. Haveria uma forma de programar um robô baseado em eventos discretos sem requerer reconhecimento de imagem?

O hardware mais utilizado em robôs é o Controladores Lógico Programáveis (CLP) que baseia sua lógica em máquinas de estado. Essa solução é amplamente utilizada na indústria, e já se provou ser eficaz em situações com intermitência de rede ou na impossibilidade de atuação de agentes humanos. Muitas tecnologias específicas já foram utilizadas para resolver problemas parecidos com este, como exemplo temos a utilização do processo de tomada de decisão probabilístico [**31**], redes neurais profundas para o processamento de imagens [**7**], redes de Petri [**35**] dentre outras. Destaca-se [**29**] que integrou a *PNRD* (Redes de Petri inseridas em base de dados RFID) e a *PNRD* invertida ou *iPNRD* para resolver problemas de planejamento para robôs em um espaço Petri. A integração *PNRD/iPNRD* produz uma arquitetura adaptativa de controle de eventos discretos onde os dados estão distribuídos tanto nos agente passivo por tags RFID(*Radio Frequency IDentification*) assim como também em agentes ativos embarcados com leitor RFID. Combinando estes componentes juntamente com o espaço de Petri (PSS) um algoritmo de busca é implementado para encontrar a sequência de transições para alcançar assim o objetivo final requerido, concluindo assim a geração de um controle de eventos discretos adaptativo. É importante destacar que essa solução é capaz de detectar automaticamente exceções ao processo e, porventura, tratar as exceções. Porém, essa abordagem se limita a problemas pequenos e não consegue solucionar problemas mais complexos. Por exemplo a integração da *PNRD/iPNRD* é eficaz para resolver um problema do mundo de blocos para 3 blocos, mas não atende 4 ou mais blocos pois o modelo do sistema em iPNRD cresce exponencialmente. Este sistema oferece um planejamento automático

utilizando computação em nuvem, mas é capaz de rodar em servidores locais em caso de queda de rede e são suportados pelos processadores existentes em chão de fábrica, pois a solução encontrada em rede de Petri é facilmente traduzida em uma máquina de estado [**15**].

O estudo de Tavares e Souza [**33**] propõe a integração entre dados PNRD/iPNRD (Redes de Petri Dentro de Base de Dados RFID e PNRD inversa) como uma solução para o problema do Mundo de Blocos, composto por três blocos, por meio de uma arquitetura de controle a eventos discretos adaptativa.Ao aplicar esta solução foram verificadas limitações em relação ao número de blocos que podem ser utilizados e a falta de aplicabilidade das informações, representados neste caso por um espaço de Petri.Além de que esta integração não propõe a possibilidade de um controle adaptativo que possa tratar de possíveis exceções e perturbações que podem ocorrer em ambiente industrial. E é importante citar que apesar de abordagem PNRD/iPNRD proporcionar uma redução do envolvimento humano, é importante ressaltar que ela se baseia em um espaço de estados fixo gerado *offline*, limitando sua aplicabilidade em certos contextos.

Na implementação de Santana [**29**] foram utilizados 3 blocos com uma aplicação direta em Arduino em que redes de Petri eram armazenadas em tags RFID que ao serem lidas por uma antena, garantiriam o armazenamento de estados já predefinidos em matrizes 3×3. Com o aumento no número de blocos neste exemplo aplicado, a configuração dos estados necessitária de uma memória de armazenamento muito maior e quanto mais blocos adicionados, um sistema mais robusto de base de dados e coleta destes seria necessário.

O uso de IoT e Computação em nuvem com planejamento *online* ainda é discutido além de uma abstração compreensiva para o usuário e a dificuldade de uma ação deliberativa [**14**] em aplicações *online* reais, o que computacionalmente difícil. O uso então de um programa que utilize a aplicação de sistemas de baixo e alto nível para integrar o processo de planejamento com a execução de atuadores é necessário para a atuação offline. Havendo a comunicação do controlador utilizado nestes sistemas com o tratamento de seus dados feito separadamente, este problema da ação deliberada no campo do planejamento automático pode ser atenuado.

O sistema que define este planejamento, ocorre ao se obter o domínio padrão como entrada para a situação momentânea poder ser descrita a cada atualização. O planejador então alimenta o controlador com dados contendo passos de atuação que serão interpretados como comandos discretos combinados. Ao executar estas ações, cada informação é tratada para que se possa obter possíveis exceções. Nesta configuração de sistema é possível encontrar exceções que ocorrem ao obter os dados e informações, denominadas exceções estáticas pois são detectadas antes das ações de movimentação, assim como exceções que podem ocorrer no momento de atuação do planejamento, chamadas de exceções dinâmicas. Muita dessas exceções podem ser tratadas e assim não precisam de ação humana direta.

Tecnologias baseadas em lógica de predicados agrupados, como, por exemplo o *Grouped Individual State Predicates* (GISP) [**5**], identificam não só cada elemento com a tag, assim como utilizam uma linguagem formal de texto para descrever domínios, e assim, definirem utilizando algum algoritmo de busca baseados nesta linguagem, os passos de atuação. O PDDL é utilizado neste contexto para solucionar problemas que definem o estado inicial e o objetivo requisitado [**14**]. Este trabalho demonstra a aplicação da abordagem *PRD* (Predicados inseridos em base de dados RFID) apresentada por [**5**]. A abordagem *PRD* é implementada em um robô linear obtendo predicados inseridos em tags e gerando os *snapshots* de problema e de objetos em PDDL que ao são utilizado como entrada para um planejador automático. Esse trabalho desenvolve tanto o módulo de percepção como também implementa o módulo de planejamento do robô linear proposto em [**32**] além de adaptação de código de Santana[**29**] onde sistemas práticos utilizando Arduino e a integração de módulos e pacotes em Python

se integram para esta aplicação. Ao se identificar cada objeto com estado definido em tag RFID utilizando-se de predicados acessados e integrados no GISP , que define todos estados agrupados de problemas (cenário do instante).Com isso pode se definir o que chamamos de PRD .A línguagem que se aplica o PRD gera um conjunto de propriedades e ferramentas, que podem ser usadas para resolver problemas e domínios de sistemas multiagente.

Ao se implementar este projeto, foram feitas várias adaptações de acordo com que foi proposto anteriormente, como, por exemplo, o sistema em baixo nível que foi implementado em Arduino, que se utilizou de uma estrutura modular que necessita de comunicação com um sistema em alto nível para que informações possam ser compartilhadas entre a utilização de cada módulo.

O sistema que será utilizado tem sua construção no ano de 2017 e possuiu várias adaptações para outros usos. A aplicação da PRD neste sistema pode conduzir uma série de melhorias em relação à velocidade e aplicabilidade em sistemas reais de produção e estoque, fazendo com que atém mesmo a estruturação de estoques reais mudem para serviços automatizados e informatizados. Além de promover uma flexibilidade e rápida investigação de peças ou produtos perdidos no ambiente de produção.

O uso de Inteligência Artificial que promove o controle adaptativo utiliza o modelo de planejamento automático referenciado na Figura 12 que executa um plano de ações de movimento e as define para um controlador que segue um baixo nível de abstração.

Os tópicos que serão definidos na discussão deste projeto são: objetivos, justificativa, fundamentação teórica, onde será realizada uma breve revisão bibliográfica acerca do tema, desenvolvimento, resultados e discussões, conclusão e trabalhos futuros e referências bibliográficas.

2 OBJETIVOS

2.1 Objetivo Geral

Projetar e implementar a PRD em um robô cartesiano para resolver o caso similar ao mundo de blocos com mais de 3 blocos, consistindo em movimentação e coleta de dados RFID de um robô cartesiano e aplicar estes módulos para solução do problema do Mundo de Blocos.

2.2 Objetivos Específicos

Considerando o objetivo geral apresentado e o desenvolvimento do projeto, os seguintes objetivos específicos podem ser destacados:

- Adaptar drivers de movimentação para cada eixo do robô cartesiano;
- Aplicar funções de leitura e escrita *RFID* conforme a abordagem *PRD*;
- Implementar a PRD no robô cartesiano utilizando Arduino(através do software);
- Criar um domínio com predicados *GISP* para problemas do Mundo dos Blocos;
- Definir módulos Python para o tratamento de texto e padrões para predicados integrado no Arduino para controle adaptativo do robô cartesiano.

3 JUSTIFICATIVA

Com a extensa utilização de serviços inteligentes em produção, é preciso adaptar novos métodos de produção e definir sistemas mais eficientes para se adequar a competição atual de mercado. Com o advento de ferramentas de Inteligência Artificial cada vez mais comuns em amplos serviços de tecnologia, em que muitos deles apresentam soluções de baixo custo e alta eficiência, além de uma utilização bastante efetiva em utilização de dados em texto, com um sistema PRD, que aplica ferramentas como estas nestes dados, é possível se obter sistemas robóticos modernos, eficientes, autorreguláveis e de custo reduzido.

Disciplinas como Eletrônica Basica e Digital, alinhadas com o uso de Python na discíplina de Sistemas Digitais, alem de conceitos de Redes Industriais e Instrumentação, auxiliaram para que os conceitos aplicados neste trabalho ganhassem forma e a implementação pudesse ser realizada.

4 FUNDAMENTAÇÃO TEÓRICA

4.1 RFID

A tecnologia RFID tem sido aplicada constantemente em várias aplicações industriais, principalmente com o uso da IoT.Consiste basicamente na "utilização de uma etiqueta plana, adesiva, de dimensões reduzidas, contendo um micro-chip em conjunto com sensores especiais e dispositivos que possibilitam a codificação e leitura dos dados contidos na mesma"[**22**].

Ao utilizar dados que necessitam de ser lidos para alguma autentificação, armazenamento de informações específicas ou outras utilizações utiliza-se a tecnologia RFID. A tecnologia de identificação por radiofrequência usa ondas de rádio para identificar objetos de forma automática, sejam seres vivos ou objetos inanimados [**7**]. No caso de informações obtidas via ondas eletromagnéticas temos o seguinte sistema: um leitor capta os dados escritos em uma Tag, assim como pode também os escrever, e isso se dá, pois o circuito da Tag é energizado, envia ou recebe estas informações, decodificadas e tratadas por uma interface de aplicação que executa estes dados.

E usando a definição citada [**22**] podemos estabelecer que existem algumas váriaveis a se levar em consideração ao se implementar esta tecnologia. Primeiramente, em relação as tags que funcionam como etiquetas planas, características como: distância de proximidade de contato e tecnologia de gravação em memória tem de ser observados para a escolha das mesmas. Já os sensores que seriam definidos, podem ser antenas como sensores especiais, sendo no processo de leitura, definidos como leitores RFID. E para estes leitores, variáveis como: Frequência de operação, tipos de cartões (ou etiquetas) suportados, taxa de transferência e dimensões tem que ser levados em consideração. Os elementos que compreendem o sistema RFID estão identificados na Figura 1.

Figura 1: Elementos de uma sistema RFID

Fonte: [**7**]

O sistema RFID apresenta uma antena que capta os dados presentes nas tags. Para o compartilhamento de dados seriais entre os dispositivos utilizamos três protocolos de comunicação frequentemente. São estes:

- **UART:**(*Universal Asynchronous Receiver/Transmitter*)Possui um transmissor de dados e um Receptor de dados que ao se conectarem por meio de conexão por fios se comunicam de forma bi-direcional em relação à transmissão de dados.
- **I2C:**Utiliza apenas um barramento compartilhado para a transmissão de dados
- **SPI:**(*Serial Peripheral Interface*)Protocolo de comunicação com sinais com uma direção fixa e definida.

Para a implementação do sistema RFID, será tradado qual sistema escolhido e também a ferramenta eletrônica que executará a tecnologia.A Figura 2 exemplifica etiquetas RFID que geralmente são utilizadas em aplicações distintas,como por exemplo, a tag adesiva, a tag chaveiro, e a tag em cartão.

Figura 2: Etiquetas RFID comumente utilizadas

Fonte: Santana,2023

4.2 Robô Cartesiano

Os robôs cartesianos, que podem também ser conhecidos como robôs lineares, são utilizados na indústria pois tem um movimento preciso e controlado em três eixos.[**36**]. Este se move em um sitema de coordenadas cartesianas, com motores em cada um dos eixos (x,y e z).[**1**] Eles são caracterizados por sua estrutura retangular ou paralelepípeda,como pode ser visto na Figura 3.

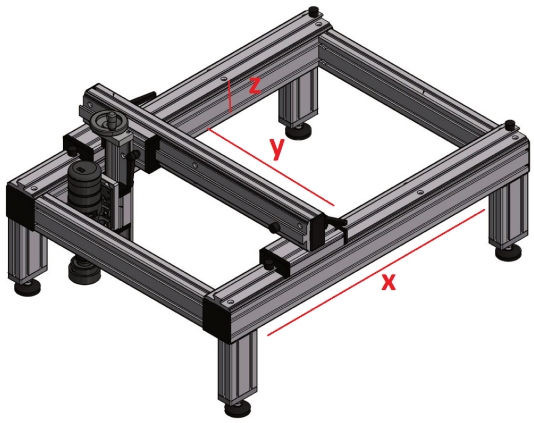

Figura 3: Modelo de robô cartesiano

Fonte: <https://www.kitotec.shop/en/kito-pt-gantry-stands.html> (Acesso em: 26/02/2024)

Como aplicações destes robôs podemos citar a Manufatura Automatizada e a Industria Eletrônica para realizar tarefas como montagem, soldagem, corte a laser e posicionamento de componentes em placas de circuito impresso (PCBs)[**4**].

4.3 Mundo de Blocos

Em aplicações industriais, onde produtos são armazenados em estoque e com isto possuem etiquetas de identificação com determinadas informações nestas, aplicações envolvendo planejamento e em robótica em geral geralmente implementam um modelo chamado de Mundo de Blocos.

Este modelo consiste de uma quantidade de blocos sobrepostos em uma plataforma, sendo que parametrizados ou não em posições distintas serão movidos de sua posição inicial, para uma posição desejada após a execução do sistema robótico.

Neste modelo, um bloco pode ser movido por vez apenas. E no caso mostrado na Figura 4, nenhum bloco pode ser empilhado em cima do outro. No mundo de blocos podemos trabalhar com qualquer disposição necessária desde que o sistema que o implementará suporte a extensão utilizada. A complexidade que será implementada levará em consideração a variação de três a cinco blocos, por limitação espacial. A distribuição utilizada foi com 5 linhas e 5 colunas em que cada posição foi definida

por meio de índices como irá ser demonstrado na Figura 5. No espaço de trabalho identificado pela Figura 4, tem se a disposição de 5 blocos em posições específicas. Estas posições estão dispostas em 25 localidades como mostrado na Figura 5.

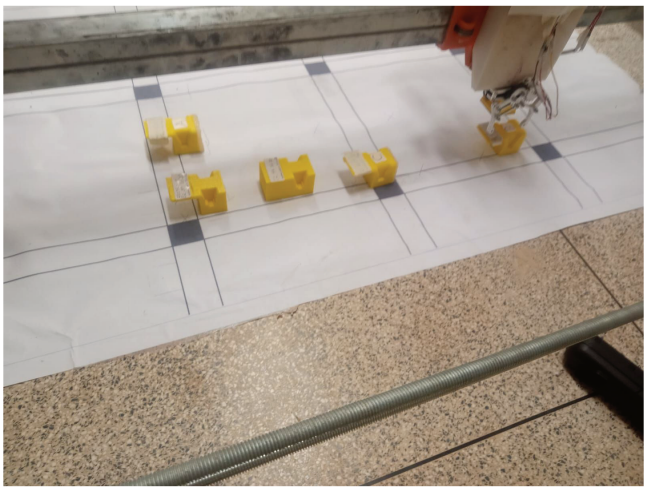

Figura 4: Fotografia do espaço de trabalho do robô simulando o Mundo de Blocos

Fonte: Acervo pessoal

21	22	23	24	25
16	17	18	19	20
11	12	13	14	15
6	7	8	9	10
1	2	3	4	5
t1	t2	t3	t4	t5

Figura 5: Mapeamento de posições no espaço de trabalho

Fonte:Acervo Pessoal

A configuração do Mundo de blocos então, não apresenta a simulação de profundidade de espaço, o que simularia o Mundo de blocos em duas dimensões de operação no espaço de trabalho. E apenas uma dimensão de ação que será definida como a manipulação dos blocos neste espaço. As regras para a movimentação nesta configuração são as seguintes:

- Nenhuma entidade pode ocupar a mesma posição,salvo um ator e um bloco;
- Apenas um bloco pode ser movido por vez por ator;
- Um bloco pode ser pego apenas se não houver outro diretamente acima;
- Um bloco pode ser colocado numa posição apenas se houver um bloco ou mesa abaixo;
- Um ator pode se mover apenas entre posições adjacentes, uma por vez;
- Todas as entidades tem identidade única.

4.4 Linguagens

4.4.1 Maquinas de Estado Finito

Maquinas de Estado Finito também chamadas de Autômatos Finitos, são modelos matemáticos que possuem um numero de estados e que podem ser utilizados para representar sistemas lógicos sendo formalizado para a resolução de problemas [**2**]. Sua modelagem de sistemas evidencia sistemas com espaço de estados discreto.

Esta abordagem se apresenta bem simples ao se construir um programa, e até mesmo na implementação deste trabalho, onde processos de identificação do espaço de estados além de um processamento sequencial pelo módulo prático executor são necessários.

A Maquina de Estado Finito como identificado na Figura 6 vai obter uma entrada de um determinado conjunto , modificar seu estado assim definir uma saída. e sequencialmente estas maquinas conduzem o processamento de informações.

Figura 6: Maquina de Estados

Fonte: Acervo Pessoal

4.4.2 Predicados

Os predicados em tecnologia são elementos que tratam do estado de um objeto/classe segundo a logica dos predicados [**23**].Esta lógica de predicados define que cada um destes elementos podem obter apenas dois estados: verdadeiro ou falso.

Com base em linguagens como o PROLOG, e modelos de representação de planejamento automático, e obtendo-se de uma base de conhecimento, uma declaração de "dúvidas"é feita para a obtenção de uma resposta procedural. Com um domínio e um problema, específicados pelo programador, uma sequencia de passos é obtida e assim ações são realizadas pela máquina com esta implementação.

Aplicações utilizando a programação logica podem ser citada como por exemplo: Alocação de elementos em estoques [**21**] e prova de teoremas matemáticos[**9**].

4.4.3 Diagramas UML

A linguagem grafica diagramática chamada de (UML) do inglês *Universal Modeling Machine* tem sua tradução como Linguagem de Modelagem Universal e esta especifica o comportamento de módulos de sistemas com conceitos baseados no paradigma de orientação a objetos **[10]**. São ferramentas que auxiliam na modelagem de domínios e sistemas.**[36]** A estrutura dos principais diagramas UML pode ser vista na Figura 7 onde se apresentam dois tipos de diagrama, onde o diagrama de comportamento se apresenta como uma demonstração da atividade e o diagrama de estrutura se apresenta como um diagrama prático de uso.

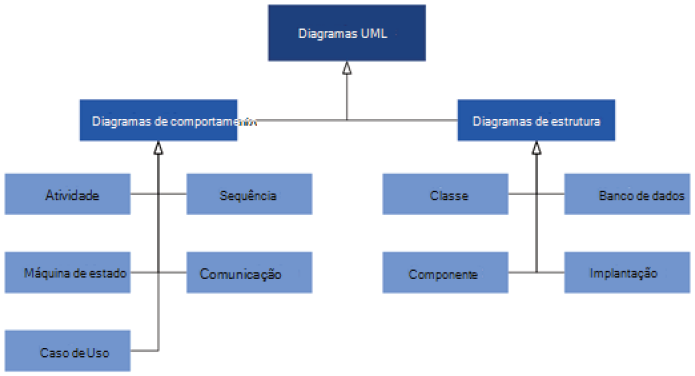

Figura 7: Estrutura de diagramas UML

Fonte: <https://support.content.office.net/pt-br/media/4500053f-e023-4185-8c57-f00ca10f2b96.png> (Acesso em: 21/02/2024).

4.4.4 PDDL e Lógica de predicados

Um sistema robótico ao ser aplicado em ambiente industrial terá seu funcionamento baseado em sistemas embarcados construídos por meio de sistemas eletrônicos. Ao se utilizar um sistema robótico eletrônico, a lógica envolvida em circuitos digitais será obtida por meio de uma união entre a ação destes circuitos acionada pelo que chamamos de lógica de predicados. Esta logica é definida como a relação entre validade ou não de sentenças lógicas. O paradigma de programação que avalia proposições (ou predicados) lógico-aritméticas é o paradigma de Programação Lógica.

Este paradigma é muito utilizado em Inteligência Artificial, pois opera utilizando os seguintes elementos:

- **Proposições:** Fatos concretos e já definidos que podem apresentar-se como "Verdade" ou "Falso".Exemplo utilizado neste trabalho: (Lugar Livre).
- **Regras de Inferência:** Regras que definem como as preposições vão ser deduzidas ou interpretadas. Usando o exemplo da preposição (Lugar Livre) temos que Livre e Lugar tem de ser relacionados por meio de uma regra que defina se este predicado se apresenta como verdadeiro ou não.
- **Busca:** Estratégia para que se possa definir o controle das inferências feitas para cada predicado.

A aplicação deste paradigma aparece em linguagens como PDDL ou até Prolog, mas para serem aplicadas em alguns planejadores, tem de usar linguagens de marcação como XML integradas com seus arquivos, por exemplo.

A linguagem PDDL possui componentes para cada tarefa executada. São estes:

- **Objetos:** Cada elemento útil de projeto.
- **Predicados:** Propriedades destes objetos, que podem ser verdadeiras ou falsas

- **Estado inicial:** Como se apresenta o estado do objeto no momento específico.
- **Especificação do Objetivo:** As propriedades do objeto desejadas.
- **Ações e Operadores:** Aplicações que operam para mudar o estado de cada predicado afim que obtendo relações verdadeiras possam seguir uma sequência de comandos.

Um sistema em PDDL necessita de dois arquivos: o que define o Domínio e o de Problema. O arquivo de domínio como demonstrado na Figura 8 define requirimentos do domínio, objetos e seus tipos, predicados e ações com suas determinadas implicações ao serem executadas. Nesta Figura, é possível ver a definição de nome por meio do marcador *:define*, requerimentos por *:requirements*, tipos por *:types*, predicados por *:predicates* e ações por *:action*. Já a Figura 9 referencia o *snapshot* do problema que evidencia seu nome,dominio, objetos e variáveis referenciadas, estado inicial e estado final. A linguagem é estruturada por meio de uma coleção de predicados entre parêntesis que ao obterem uma condição de verdadeiro, ou falso definem ações que ao serem tratadas em algum planejador automático obtém uma sequência de passos a serem executados para o objetivo final ser concluído como demonstrado nas Figuras 8 e 9. A Figura 10 determina o resultado do plano gerado. Os resultados geralmente definem os estados iniciais assim como nós, que seriam a configuração virtual destes estados, custos e cada ação planejada.

```
(define (domain iRoPro)
 (:requirements :strips :typing)
 (:types
    element
    position - element
    object - element
    cube - object
    base - object
    roof - object )
 (:predicates
    (clear ?e - element)
    (thin ?o - object)
    (flat ?e - element)
    (on ?o - object ?e - element)
    (stackable ?o - object ?e - element)
 (:action move
  :parameters (?o - object ?A - position ?B - position)
  :precondition (and (on ?o ?A) (clear ?o) (clear ?B)
  :effect (and (on ?o ?B) (clear ?A)
                    not(on ?o ?A) not(clear ?B))
)
```

Figura 8: Padrão PDDL para domínio de sistema

Fonte: <https://www.researchgate.net/figure/Example-of-a-planning-domain-in-PDDL_fig4_350638824> (Acessado em : 22/02/2024).

```
(define (problem PROBLEM_NAME)
  (:domain DOMAIN_NAME)
  (:objects OBJ1 OBJ2 ... OBJ_N)
  (:init ATOM1 ATOM2 ... ATOM_N)
  (:goal CONDITION_FORMULA)
  )
```

Figura 9: Estrutura para problema

Fonte: <https://users.cecs.anu.edu.au/ patrik/pddlman/writing.html> (Acessado em : 22/02/2024).

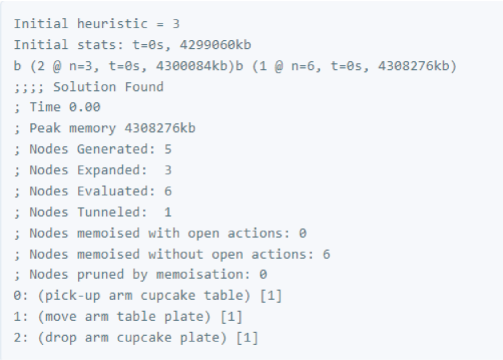

Figura 10: Exemplo de um plano gerado

Fonte: <https://fareskalaboud.github.io/LearnPDDL/> (Acessado em : 22/02/2024).

4.4.5 Módulos e pacotes Python

A linguagem Python possui determinadas bibliotecas ou pacotes de programas variados, que podem ser acessados livremente por meio de serviços de instalação, como, por exemplo, o **PYPI**. Este serviço garante que módulos ou pacotes produzidos por pessoas no mundo todo, possam ser distribuídos por meio de livre acesso. E ao se definir qual será o modo em que a informação será disposta, existem variados pacotes utilizados.Por exemplo:

- **Numpy:** Pacote que garante o controle da disposição matemática de determinados dados. Manipulações matemáticas envolvendo matrizes, métodos matemáticos mais complexos, por exemplo, podem ser feitos por meio de métodos e atributos das classes do Numpy.

- **Pandas:** Garante a manipulação de arquivos separados por vírgulas, que simulam a disposição tabular de informações. Este pacote garante a manipulação por meio de linguagem Python destes arquivos, executando funções feitas em Excel.

- **RE:** O pacote **RE**[**27**] garante funções de coleta de padrões de mensagem, para a filtragem de determinadas informações requeridas.

Além dos pacotes mostrados, existem pacotes para a manipulação e construção de arquivos de domínio e problema PDDL, e a utilização de pacotes que garantem a automação de codificação em Terminais, sejam estes de Windows, Linux ou Mac.

4.4.6 RegEx

Expressões Regulares (RegEx) são padrões utilizados para encontrar e manipular padrões de texto. Essas sequências de caracteres são especialmente úteis para busca, validação e manipulação de strings de texto em uma ampla variedade de aplicações [**11**].

O princípal uso das expressões regulares se dá na criação de padrões complexos de busca e substituição de texto. Estas podem representar simples correspondencias exatas até padrões complexos de busca.

Estas expressões geralmente são utilizadas em serviços web na validação de entrada de texto e dados, manipulação de entrada de dados que é muito presente em formulários da web por exemplo, na extração de informações e na manipulação de texto em massa.

A Figura 11 mostra um excemplo de padrão RegEx que tende a definir a extração de mensagens PRD filtrando apenas os predicados *:init*. A função **re.findall** guarda estes valores em uma lista.

```
#Criar relação da posição com o bloco
blocosIniciais = re.findall(":init\(on \w \w+\)",txt1)
```

Figura 11: Exemplo de uso da função **re.findall**

4.5 Planejamento Automatico

O Planejamento Automático pode ser definido como a geração de uma sequência de ações já definidos(planos), utilizando a abstração de modelos de domínio e problema, para a identificação,utilizando estruturas de busca, de um plano de estados intermediários que leva o sistema do estado atual para um estado objetivo solicitado. O domínio é a especificação lógica para ações ou funções específicas. O problema é a identificação de todas as características atuais do sistema para a identificação do plano.

Os planejadores podem encontrar soluções para problemas complexos desde que a estruturação dos sistemas, tanto simulados quanto físicos não contenha situações não programadas para a atuação. Em aplicações de robótica, o uso de planejadores automaticos vem ganhando espaço, e como exemplo temos a implementação de planejamento de ação baseado em PDDL no Sistema Operativo Robótico(ROS).

A escolha do tipo de busca correta além de heurísticas eficientes podem garantir planos mamis efetivos, um custo computacional menor da busca, até se necessário, a garantia de planejamento ótimo.

O uso de Inteligência Artificial que promove o controle adaptativo utiliza o modelo de planejamento automático referenciado na Figura 12 que executa um plano de ações de movimento e as define para um controlador que segue um baixo nível de abstração

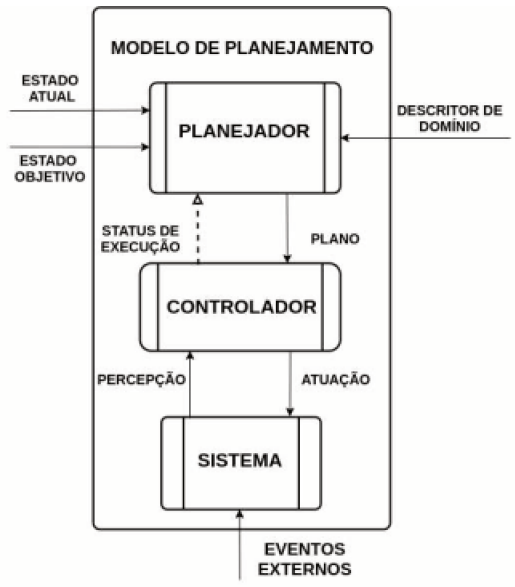

Figura 12: Modelo clássico de planejador automático

Fonte: [**32**]

4.5.1 Algorítmos de Busca

Os Algoritmos de Busca, são tecnicas para que se possa percorrer os variados estados de um grafo.Estas técnicas usualmente são determinísticas e baseadas em

modelo. A busca geralmente é feita seguindo a direção do estado atual até o estado objetivo, ou utilizando a direção contrária, que seria partindo do estado objetivo para retornar a um estado solicitado anteriormente.

Algumas propriedadades são listadas em [**28**] e podem ser definidas abaixo.

- Completa: O algoritmo de diz completo ao encontrar qualquer solução alcançável em um tempo infinito.
- Ótima: Um algoritmo se diz ótimo ao se encontrar um plano em custo mínimo.
- Complexidade de tempo e espaço: Ao estes algoritmos estarem acolados em seu problema e subespecificações ele possui complexidade de tempo e espaço.

Para realizar uma busca, primeiramente temos que específicar qual seria a característica de seu algoritmo. Ao se obter domínios e problemas correlacionados, podemos dizer que identificar o conhecimento base é necessário. Algoritmos que não dependem destas informações, são chamados de algorítmos cegos.São algoritmos que percorrem os estados, apenas até obter o estado objetivo.

Os principais algoritmos que tem este comportamento são : buscas em largura e buscas em profundidade.De forma breve, a busca em largura vai percorrer os nós do grafo até a maxima profundidade deste ramo, até que não haja mais possibilidades, obtendo assim a solução se dado a este um tempo infinito e como sua implementação é por meio de uma fila que expande novos estados que substituem os estados antigos no final desta.

Já a busca em profundidade, que por ser implementada por uma fila que expande novos estados para o topo desta não há garantias de plano ótimo ou mesmo de completude, se por exemplo obtermos uma profundidade infinita.

Os algoritmos informados, já funcionam de forma contrária aos algoritmos listados anteriormente pois estes possuem conhecimento advindo das heurísticas, que são uma quantificação da probabilidade da solução ser adquirida.Cada um dos algoritmos informados tem uma politica para suas buscas, e aliados ao valor da heurísticas calculam qual será o proximo nó percorrido. São exemplos de algoritmos informados:

- **Busca Gulosa**: A abordagem da busca gulosa geralmente avança os estados sem retroceder em nós já visitados. Sua fórmula define a função de custo f(n) (Equação 4.1), que é derivada exclusivamente da função h(n). Ao prossegui em estados mais profundas, essa função tende a se deslocar em direção aos nós mais profundos, tornando-se assim incompleta e subótima.

$$f(n) = h(n) \tag{4.1}$$

- **Busca A***: Nesta busca, há a adição de um termo de custo real até a posição g(n), adicionando assim um efeito de memória que se estabelece em cada estado percorrido conforme Equação 4.2. Este já se apresenta como ótimo e completo para heurísticas admíssiveis. A complexidade de tempo e espaço deste algoritmo se apresenta inferior a outros citados.

$$f(n) = g(n) + h(n) \tag{4.2}$$

4.5.2 Heurísticas

Heurísticas podem ser definidas como funções ou informações específicas do domínio utilizado para guiar um processo de busca. Estas funções são como expectativas tiradas conforme a proximidade da solução deste, ou seja, uma avaliação de um nó específico e com as informações do estado deste, predizer a qualidade de nós sucessores.

Estas podem ser geradas, conforme o domínio ou com as características dos problemas. Como exemplos de heurísticas temos:

- Heurística de distância euclidiana: Calcula a distância vetorial dos nós e busca o menor resultado
- Heurística de distância de Manhattan: Procura a maior distância entre nós[**18**]

Com a presença de custos variados de ações do no plano, ou características de modelagem de domínio, as heurísticas não se comportam bem desde que estas características exerçam alguma influência no cálculo destas.[**6**]

Logo características especiais que ferem o uso da heurística alvo devem ser evitadas.

4.6 Planejador Automático PDDL4J

Planejadores automáticos específicos podem ser utilizados em diversas aplicações e utilizar determinadas buscas para garantir o melhor plano, com a sequência de ações garantindo o menor custo possível.O PDDL foi originalmente desenvolvido em 1998. Teve como objetivo a melhora em comunicação dos resultados das pesquisas desencadeando assim uma explosão em desempensho , expressividade e robustez aos sistemas de planejamento.Este planejador possui a caracteristica de realizar avaliações empiricas de sistema de planejamento com domínios de planejamento adotados[**25**]. Com o PDDL sendo uma linguagem padrão para descrever domínios de planejamento, este planejador oferece um uso simples com apenas uma linha de comando para oferecer plano,custos e funcionamento do planejamento automático.

Integrado com a IDE VSCode com extensões para visualização específica para testes envolvendo domínios de planejamento PDDL temos a configuração de planos como mostrado na Figura 13 que representa o plano de ações deste trabalho em que a ação *pickBlock* define o movimento do robô linear por meio de dois eixos e a ação *stack* define a ação da garra.

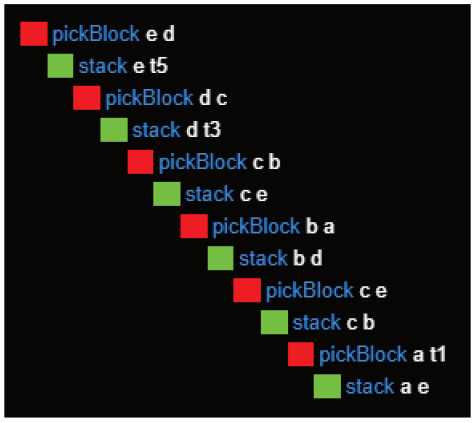

Figura 13: Plano identificado por meio de extensão no VSCode

Fonte: Arquivo Pessoal

4.7 PRD

Em indústrias em geral, tem-se a necessidade de registrar, equipamentos, peças e outros materiais necessários no processo produtivo. O tipo de registro escolhido necessitaria de um padrão já definido de sua estrutura de informação. Padrões de caracteres distintos podem ser aplicados em cada tecnologia de identificação, sendo estes com números, letras e outros símbolos necessários. Mas ao estabelecer informações mais específicas sobre os principais elementos, ou até para que sistemas inteligentes possam identificar qual o setor posicionar estes elementos identificáveis, é necessário um grande trabalho de logística que ao ser aplicado demanda tempo significativo em qualquer setor de armazenamento.

Para que estes processos possam ter uma definição mais apurada, é necessário se trabalhar com linguagens de marcação de dados.

4.7.1 Marcação de dados

Linguagens de marcação em definição: "são um conjunto de sinais e códigos aplicados a um texto ou a dados para definir formatos, maneiras de exibição e padrões"[**31**]. Com a utilização destas linguagens os processos podem utilizar marcadores ou tags que ao serem identificados, trazem definições de como o conteúdo deve ser exibido.

Como exemplos de linguagens de marcação, temos amplamente utilizada por serviços web, as linguagens JSON(*Javascript Object Notation*) e XML(*eXtensible Hypertext Markup Language*). Na representação da PRD tanto o JSON quanto o XML não possuem utilidade por alguns fatores:

- Tem seu trabalho definido por chave e valor, limitando a possibilidade de mensagens com uma liberdade de armazenamento de texto
- O JSON possui seu uso mais vinculado com a web, necessitando de pacotes Python específicos que podem não se integrar diretamente com a máquina física
- O XML é uma linguagem bastante verbosa, o que causaria uma utilização extensa em relação à memória da tag, além de também necessitar de muitos pacotes Python para o tratamento.

4.7.2 Estrutura PRD

A notação PRD e seus dados auxiliares é representada por elementos chamados marcadores.Cada marcador possui a notação *:marcador*. Cada marcador possui um ou mais predicados, que definem o estado ou demais características como, por exemplo, objetivo final ou prioridade que o elemento identificado irá ter. Na Figura 1, por exemplo, podemos ver o padrão que será utilizado neste projeto que define o marcador *:init* define o estado atual onde (blocoA acima blocoB), ou seja, utilizando a preposição *on*, definimos que o bloco A está acima do bloco B neste estado.

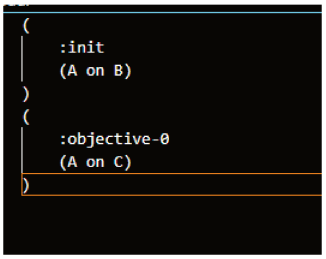

Figura 14: Estrutura de dados utilizada em sistema.

Fonte: Arquivo Pessoal

Neste exemplo da Figura 14, temos dois marcadores: *:init* que define todos os predicados que definem a posição inicial. No exemplo temos apenas um predicado

inicial que define que o Bloco A está acima(em inglês *on*) do Bloco B. O segundo marcador deste exemplo é o *:objective-0* que define o predicado do objetivo final, que define o Bloco A acima do Bloco C.

Ao trabalhar com esta estrutura de dados, aplicada em cada um dos blocos, podemos ter uma sequência de posições genéricas em que cada bloco está e onde estará ao final de toda a movimentação ou atuação do módulo robótico. O modelo que integra a varredura com a definição de planejamento automático pode ser demonstrado na Figura 15, neste modelo podemos observar como funcionam os módulos de processamento do sistema PRD e a sua implementação em módulos robóticos.

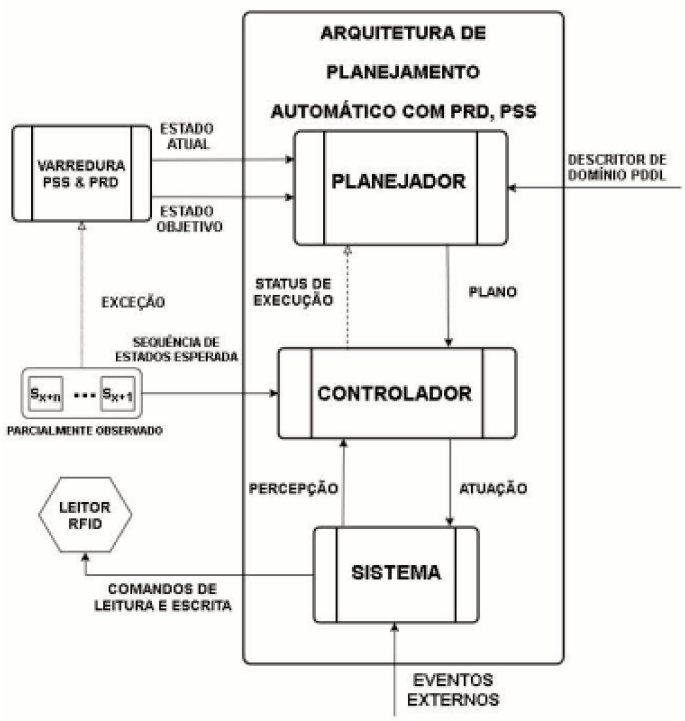

Figura 15: Módulos de Percepção e Planejamento integrados

Fonte: [**32**]

4.8 Sistema Embarcado e Robô Cartesiano

O sistema embarcado utilizado apresenta um sistema composto de:

- Robô cartesiano
- Arduino Mega
- Shield RAMPS 1.4
- Arduino UNO

Para implementar o sistema de forma que ele possa garantir o movimento e execução em tempo real, estas ferramentas têm de estar integradas em montagem, o que vai ser definido em capítulo posterior. Como cita [**22**], "Um robô cartesiano é uma máquina cujos três principais eixos de controle são lineares e perpendiculares entre si.".

Utilizando este robô em laboratório, se eliminam variáveis relacionadas a rotações variadas, fazendo o cálculo matemático e adaptações aritméticas não tanto necessárias quanto a implementação dos processos de forma prática. Ao obter sistemas robóticos com motores e articulações que possuem um sistema dinâmico mais complexo, a simulação dinâmica de trajetória é amplamente recomendada.

Essa configuração estrutural é utilizada em equipamentos de diversas áreas trazendo segurança e qualidade. Cada eixo pode ser entendido como uma junta prismática[**22**]. A Figura 16 representa a estrutura de montagem e movimentação dos motores do robô cartesiano, sendo os motores representados por meio das estruturas em vermelho.

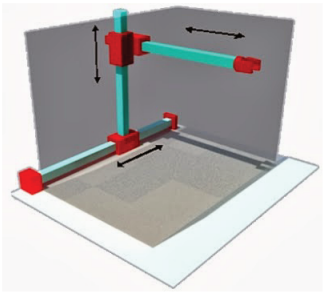

Figura 16: Estrutura de robô cartesiano utilizado.

Fonte: <https://1.bp.blogspot.com/-w5CHaLpLzhs/Uv1DXhtkN3I/AAAAAAAAGJI/R6y7hmETN
(Acessado em: 22/02/2024).

Normalmente o controle desse tipo de robô é simples, uma vez que a movimentação em cada eixo representa uma relação direta no movimento do elemento terminal do robô.

4.9 Arduino Mega

O Arduino Mega possui a finalidade de fazer a comunicação entre o leitor RFID acoplado à garra robótica e o computador, que envia ao *software* os dados contidos nas *tags* RFID, para que o *software* possa utilizar estes dados para calcular os movimentos que a garra robótica deverá fazer, a fim de movimentar cada bloco. O Arduino Mega possui uma comunicação USB chamada de Serial. A Serial, ao ser conectada em uma Porta COM, exibe as informações passadas por meio de escrita de dados. Esta escrita pode ser feita pela IDE do Aduino, além de poder ser feita por outros programas.Esta exibição, também pode ser chamada de comunicação serial de dados da porta USB. O Arduino Uno está identificado na Figura 17.

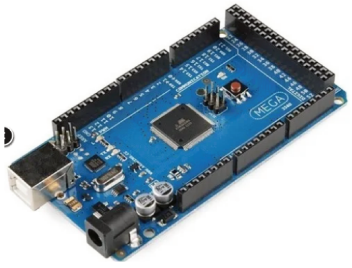

Figura 17: Arduino Mega

Fonte: <https://www.eletrogate.com/mega-2560-r3-cabo-usb-para-arduino> (Acessado em: 22/02/2024).

4.10 RAMPS 1.4

A *shield* RAMPS 1.4 é quem faz o gerenciamento energético e das informações enviadas pelo Arduino Mega através dos pinos de controle: Se o Arduino recebe o comando para mover um eixo XYZ do robô, é a RAMPS quem acionará o driver do respectivo motor (e consequentemente o motor em si).A Figura 18 demonstra a *shield* com suas conexões.

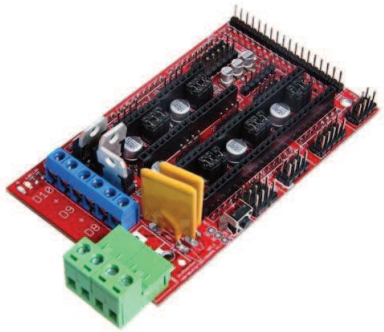

Figura 18: Módulo *shield* RAMPS 1.4.

Fonte: <https://proesi.com.br/modulo-shield-ramps-1-4.html> (Acessado em: (22/02/2024).

4.11 Leitor RFID PN532

O leitor RFID se apresenta como principal antena para a identificação de dados. A transmissão dos dados é feito por meio de conversão das informações binarias em tag para texto. Na Figura 19 podemos identificar diferentes conexões, sendo o *Switch* responsável por identificar qual protocolo de comunicação sé utilizado para a transferência dos dados e as conexões correspondentes aos protocolos I2C,HSU e SPI. Cada protocolo define a biblioteca utilizada e neste trabalho tanto o protocolo I2C quanto o protocolo HSU serão demonstrados na proxima seção.

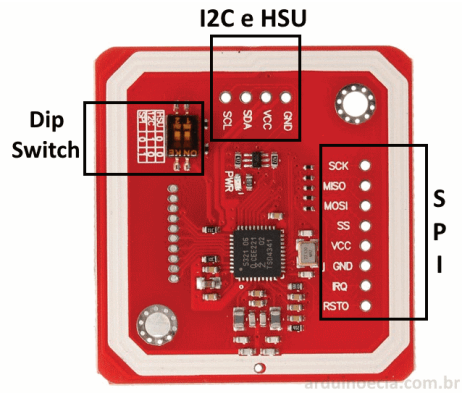

Figura 19: Leitor RFID PN532

Fonte : <https://www.arduinoecia.com.br/modulo-pn532-nfc-rfid-arduino/> (Acessado em: (22/02/2024).

5 METODOLOGIA E DESENVOLVIMENTO

Este trabalho tem como objetivo a implementação do modelo PRD ja proposto por [**32**] e [**32**], onde o sistema aplica os três módulos da estrutura completa da PRD de forma prática em um robô linear implementado no Campus Glória da Universidade Federal de Uberlândia.

Esta seção é dividida em especificação de modelo e arquitetura utilizada para implementação do sistema PRD, além de estabelecimento de ferramentas e materiais utilizados para a construção dos serviços aplicados.

Neste modelo, o sistema seria a aplicação de três módulos essenciais como definido na Figura 20. Estes três modulos conduzem estruturas de Sensoriamento, Planejamento Automático e Execução de Drivers Robóticos. Cada uma delas pode ser definida como:

- **Percepção:** Etapa em que o módulo robótico aplica uma varredura, com o objetivo de localizar blocos no espaço de trabalho definido e gerar um *snapshot*, ou seja, uma captura em tempo real, do estado atual do sistema.

- **Planejamento:** Etapa que define, utilizando um planejador integrado ao módulo fisico, o plano de ações que o modulo robotico deve executar para chegar em um determinado estado objetivo.

- **Execução:** Etapa em que, ao receber o plano gerado pelo planejador automático, o sistema converte os dados recebidos em ações de movimentação dos atuadores deste sistema robótico, executando assim o plano automáticamente.

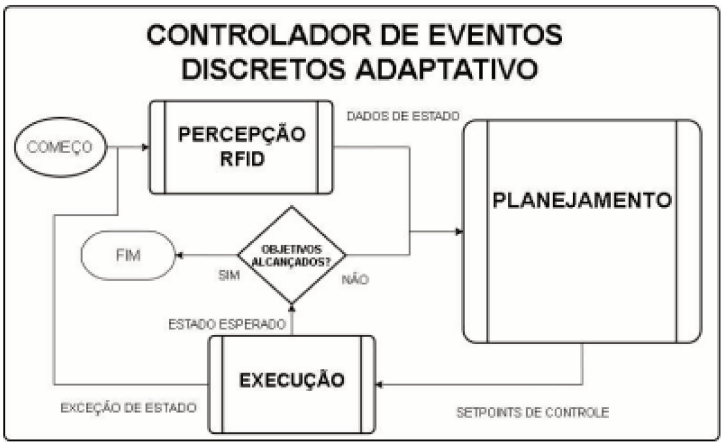

Figura 20: Modelo do Controlador de Eventos Discretos Adaptativo

Fonte: [**32**].

5.1 Percepção

O modulo da percepção tem como atividade, a identificação de toda configuração física. No Mundo de Blocos, esta configuração física compreende na disposição identificada na Figura 5 alem das informações presentes nas tags. A identificação feita por RFID garante a uníca identificação do bloco devido ao ID estabelecido em cada tag. A Figura 21 estabelece qual seria a identificação no módulo RFID.

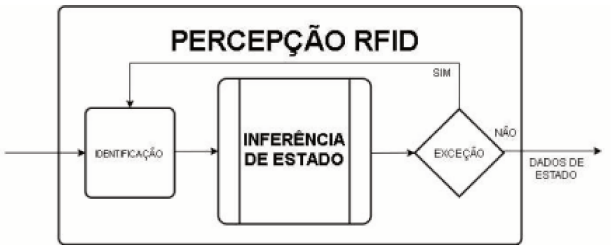

Figura 21: Módulo de Percepção RFID

Fonte: [**32**].

Este módulo pode se dividir em três etapas: Identificação por Leitor RFID, Inferência de estado por meio de comunicação Serial, e Verificação de Exceções Estáticas.

Para a etapa de Identificação do Leitor RFID, temos a leitura da tag e a obtenção desta em texto. A tag possui a configuração da Figura 22 onde nesta estão a identificação(ID) e o *Payload* que são todos os dados escritos na tag.

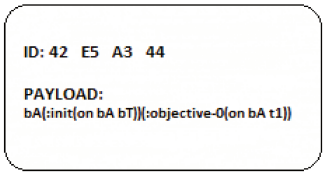

Figura 22: Módulo de Percepção RFID

Fonte: Acervo Pessoal

Ao identificar todas os blocos necessários, o sistema garante uma inferência de estado, também chamado de *snapshot* onde todos os blocos podem ser identificados, além de suas posições físicas. Esta inferência de estado pode ser determinada por meio da captura de informações em uma comunicação serial. A comunicação Serial dita por texto, todo o processo prático realizado fisicamente. A Figura 23identifica uma tela do IDE do Arduino identificando a comunicação serial e ilustrativamente, identifica possíveis estados.

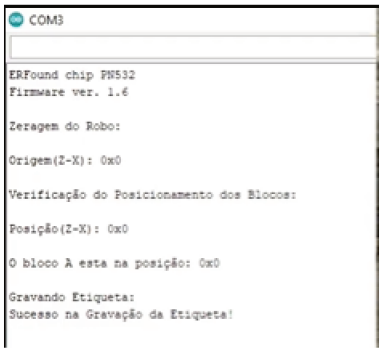

Figura 23: Tela de Exibição da Comunicação Serial

Fonte: Acervo Pessoal

Com a captura dos caracteres obtidos da comunicação serial, feita no intervalo de tempo em que a Percepção ocorre, obteremos o *snapshot*.

Com o *snapshot* obtido por inferência de estado, o processo de tratamento de exceções busca variadas exceções. Com a possibilidade de se encontrar exceções de percepção e também de execução, classificaremos estas exceções em: Estática e Dinâmicas. As exceções estáticas são aquelas que aparecem no módulo de Percepção.

Destas Exceções podemos destacar as seguintes:

- **Exceções de Posicionamento:** Estas exceções acontecem quando a posição real da maquina não está em conformidade com a posição da informação na PRD.
- **Exceções de *Snapshot*:** Esta exceção ocorre ao obter no *snapshot*, predicados, tanto de *:init* quanto de *:objective-0* que não possam executar, como por exemplo, informações vazias em predicados *:init*.

5.2 Planejamento

O modulo de Planejamento, só pode ser executado, ao obter uma confirmação da inexistência de exceções estáticas.Neste módulo, o planejador recebe informações do PRD e PSS, que estão todas no *snapshot*. O PSS pode ser definido como o espaço de estados definido pela Figura 5. A Figura 24 exemplifica a estrutura do módulo que possui dois estados de planejamento. O primeiro módulo planeja as transições discretas, ou seja, o plano de ações discretas que ao ser executado faz com que o estado obtido, seja o estado objetivo. O segundo planeja a conversão de texto para ações em baixo nível, transferindo assim os comandos para o controlador conforme demonstrado na Figura 24.

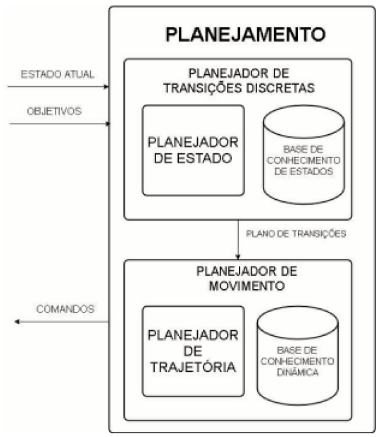

Figura 24: Módulo de Planejamento simplificado

Fonte: **[32]**

O planejamento automático de transições é definido por meio da atuação de um planejador que recebe tanto domínio quanto problema, e identifica todas as ações necessárias para o estado objetivo ser alcançado. O planejador gera uma mensagem em texto contendo todos as ações em sequência, e o planejador de movimento conectado com a comunicação serial, gera códigos também em texto, porém de forma resumida e compactada como mostrado no exemplo da Figura 25.

A Figura 25 determina como o planejamento será executado, com o *snapshot* sendo entregue para o planejador de transições que irá gerar um plano assim como o da Figura 13 que ao ser definido em um arquivo e entregue como entrada para o planejador de movimentos, todos os movimentos serão definidos pela conversão em comandos de alto nível em baixo nível. Obtendo o fim do plano, este módulo termina.

Figura 25: Fluxograma do funcionamento do módulo de Planejamento

Fonte: Acervo Pessoal

5.3 Execução

O módulo de execução busca a atualização da PRD nas tags e o Tratamento de Exceções Dinâmicas. A Figura 26 demonstra o módulo e seus passos de execução, onde de acordo com a base de conhecimento fornecida pelo IDE controladora da comunicação Serial do arduino, temos a execução do comando enviado pelo planejador por meio da tradução deste comando para movimentações físicas. Após se movimentar, uma nova inspeção RFID é feita para a checagem de alterações de configuração de peças no espaço de estados.

A única Exceção Dinâmica que pode ocorrer, ao se levar em consideração a configuração dos blocos no espaço físico é a exceção de alteração externa deliberada da posição do Bloco. Se houver alguma manipulação de blocos, seja retirando o bloco de posição, ou algo relacionado à sua identificação, a exceção é ativada. Essa Exceção é demonstrada na Figura 27 onde o bloco foi removido deliberadamente por agente externo.

Figura 26: Módulo de Execução

Fonte: Acervo Pessoal

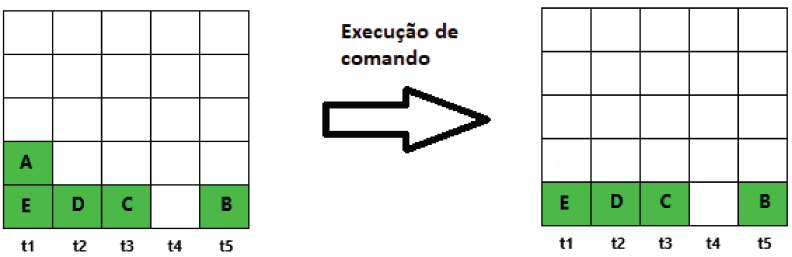

Figura 27: Exceção Dinâmica

Fonte: Acervo Pessoal

Com a exceção dinâmica devidamente tratada, há uma nova inspeção para a atualização do estado. Na próxima seção este processo será observado em sua completude.

5.4 Elementos e Materias para a Implementação da PRD

Neste trabalho a intensa integração de aplicativos e softwares, uma estruturação de sentido para os predicados, número de blocos e informações escritas em tags assim como o robô linear utilizado para a implementação da PRD tem que ser definidos para a execução completa.Eis os elementos listados:

- IDE Visual Studio Code: Utilizado para a conexão entre programa embarcado no Arduino e conexões de informações na comunicação Serial.
- IDE Arduino: Programa para a escrita e carregamento de programa embarcado
- Computador pessoal
- Robô linear com motores de passo como atuadores e Sensores Fim de curso para verificação de limítes físicos
- Módulos Python para escrita de arquivos PDDL e obtenção de exceções por arquivos de texto
- Fonte de alimentação do sistema
- Conjunto Ramps 1.4/Arduino Mega

Estes elementos serão demonstrados nas subseções abaixo.

5.4.1 IDE Visual Studio Code

Ao se desenvolver sistemas que utilizam de linguagens de programação diversas, IDEs são utilizados. IDE ou Ambiente de Desenvolvimento Integrado, é um software que reúne ferramentas de edição e compilação de aplicações, contendo este uma interface de usuário gráfica(GUI).

O IDE do Visual Studio Code foi desenvolvido pela Microsoft e se apresenta uma boa escolha ao se procurar um software de desenvolvimento leve e com extensões de livre acesso feito por desenvolvedores do mundo todo. Além de ferramentas que facilitam o versionamento de código, como, por exemplo, o GitHub. A Figura 28 demonstra a interface do Visual Studio Code. Os códigos deste trabalho posssuem todas suas versões em GitHub em que todo o processo pode ser observado.

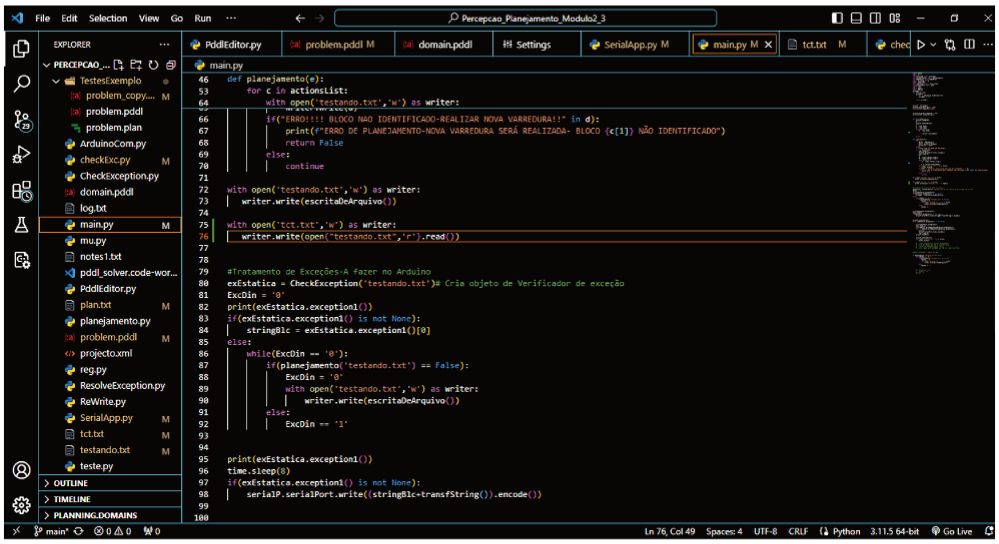

Figura 28: Interface do Visual Studio Code

Fonte: Acervo Pessoal

5.4.2 IDE Arduino

A IDE do Arduino, diferentemente da do VS Code, apenas tem suporte para uma linguagem integrada. Esta linguagem é composta por bibliotecas em C++, e atuação em linguagem C e possui duas funções específicas. Ao se integrar bibliotecas tanto da própria IDE quanto externas, diferentes elementos podem ser controlados. Neste trabalho há a inserção das bibliotecas **AcelStepper**,que oferece métodos para o controle de motores de passo, e **NfcAdapter** que possui métodos para a Leitura,Escrita e Formatação de tags RFID.

5.4.3 Extensão PDDL

A Extensão do VS Code [**16**] que pode gerar de forma gráfica, todos os estados, e a coordenação de um melhor funcionamento dos planejadores automáticos foi a Extensão PDDL. Essa extensão possui ferramentas de simulação de codigos de planejadores diretamente pelo terminal. A Figura 7 exemplifica de forma gráfica o plano gerado, e as Figuras 29 e 30, demonstram estados de predicados específicos. A Figura **??** define a plataforma do VS Code que estrutura o problema utilizado um modelo em tabela, para verificar se os predicados são verdadeiros e logo depois executar as ações que melhor definem estes estados.

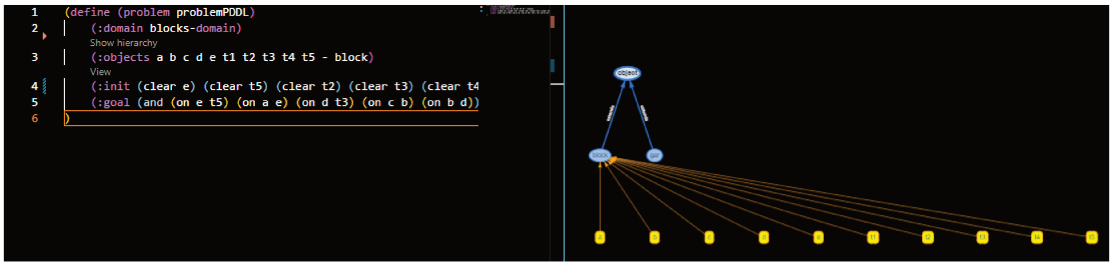

Figura 29: Interface do Visual Studio Code

Fonte: Acervo Pessoal

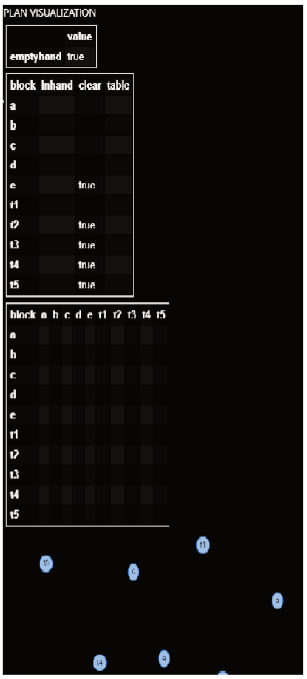

Figura 30: Interface do Visual Studio Code

Fonte: Acervo Pessoal

5.4.4 Módulos Python utilizados

Aliado ao IDE do VS Code algumas extensões podem auxiliar o processo de programação com a aplicação de algumas ferramentas, sendo uma delas o Terminal. Este terminal funciona de forma parecida aos terminais integrados a Windows e Linux. O terminal do VS Code possui códigos de download de pacotes Python utilizando o PYPI. Pacotes utilizados neste trabalhos, como por exemplo Pyserial[**3**] e PDDL 0.4.0[**8**].

O módulo Pyserial tem por objetivo a conexão com portas conectadas em um computador pessoal, e ao se conectar com uma placa na porta USB, este módulo irá estabelecer uma conexão.Porem, esta conexão terá atributos que terão de ser especificados como por exemplo: Portas de conexão USB , tempo de *timeout*(tempo para o envio de dados por conexão Serial) e *baudrate* que é o número de vezes em que um sinal em uma porta de comunicação muda sus estado.

Já o módulo PDDL 0.4.0 define métodos e atributos para se editar e configurar arquivos pddl, sejam arquivos de domínio ou de problema.Com esse módulo é possível realizar a Re-edição adaptativa em relação ao estado atual, conduzindo assim a edição do problema conforme o *snapshot*.

5.4.5 Robô Linear

O robô utilizado neste trabalho foi desenvolvido em trabalhos prévios. Foi confeccionado no trabalho de [**24**] ,aperfeiçoado e utilizado no trabalho de [**35**] e possui o código-base de funcionamento definido em [**29**]. Suas coordenadas podem ser observadas na Figura, e o movimento das coordenadas x e z é dado por motores de passo ligados por correias de transmissão que executam estes movimentos.O movimento da coordenada y é dado pelo movimento do motor de passo em sua direção.

Na Figura 5 podemos observar a classificação de cada direção em relação ao posicionamento de cada bloco. E em adição, podemos observar na Figura 31 como, físicamente o espaço de trabalho é dividido.

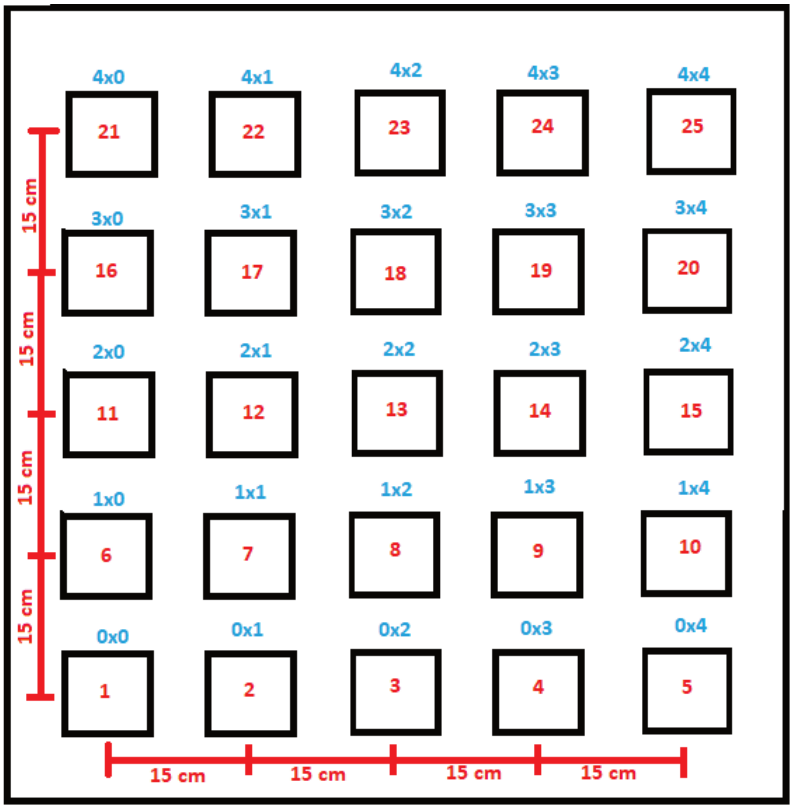

Figura 31: Robô linear utilizado no projeto e direções de movimentação

Fonte: [**29**]

A limitação da movimentação e a direção desta, é definida por sensores fim de curso, que ao serem acionados garantem um sinal que pode ser utilizado para a movimentação parar ou para definir a referência inicial utilizada. Sua alimentação e o controle da direção dos seus motores são definidos respectivamente pela fonte de alimentação e o conjunto embarcado de placas Ramps 1.4 e Arduino Mega como demonstrado nas subseções seguintes.

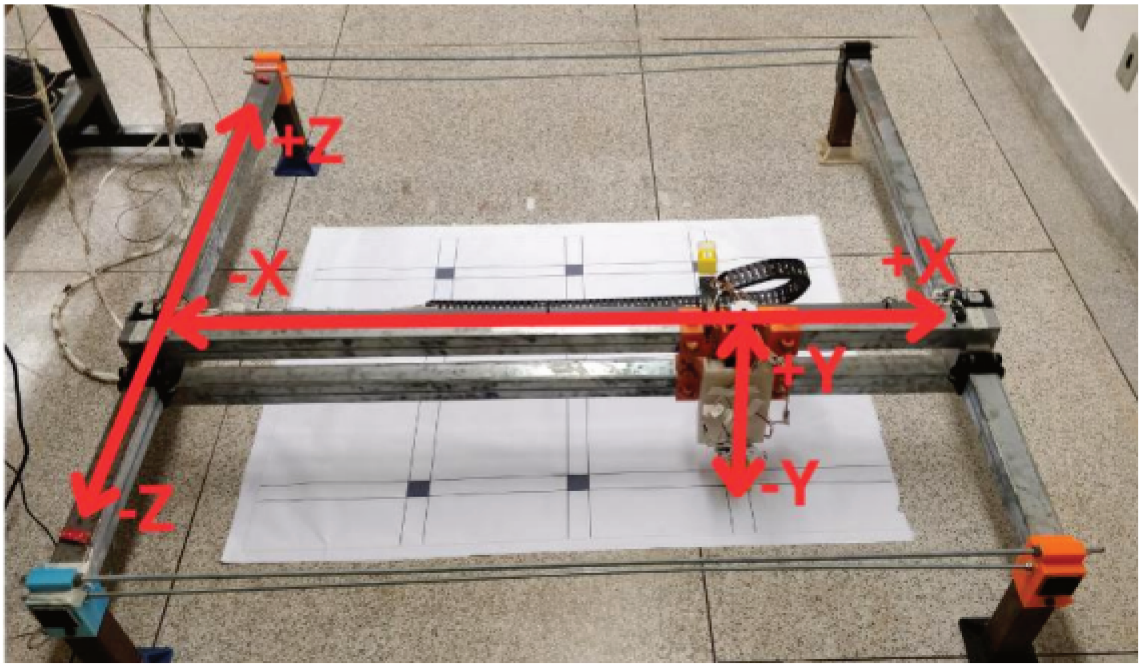

Figura 32: Robô linear utilizado no projeto e direções de movimentação

Fonte: [**29**]

5.4.6 Fonte de alimentação

A alimentação do sistema automático tem que ser dado por uma fonte de alimentação de corrente contínua, que converte a saida da rede elétrica que possui geralmente de 110 V a 220 V em saídas de 12 V para a Ramps 1.4 [**29**],a fonte pode ser vista na Figura 33.

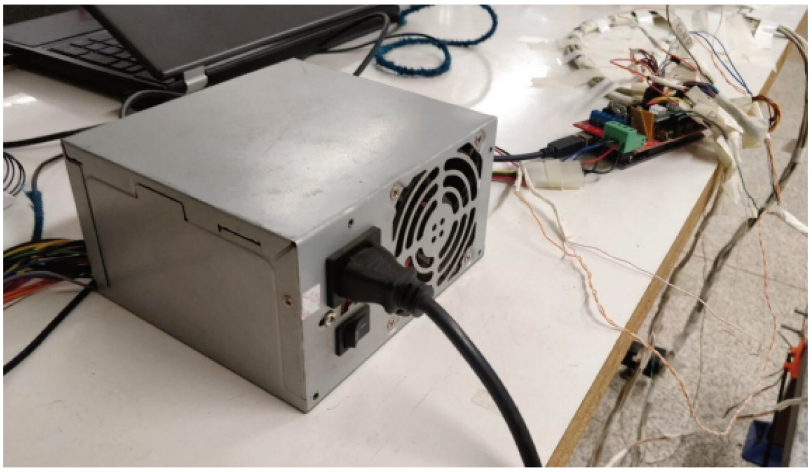

Figura 33: Fonte de Alimentação utilizada no trabalho

Fonte: [**29**]

5.4.7 Conjunto Ramps 1.4/Arduino Mega

Para ser possível haver o controle de drivers em motores de passo, o Arduino Mega necessita de placas auxiliares que os coordenam e delimita funções características para suas conexões, Por exemplo, existem conexões para cada motor de passo, garantindo assim o controle de movimentação de cada direção. A Figura 34 mostra esta conexão que com a conexão ao computador, pode garantir com que o programa seja carregado diretamente. Estas conexões podem ser demonstradas na Figura 35.

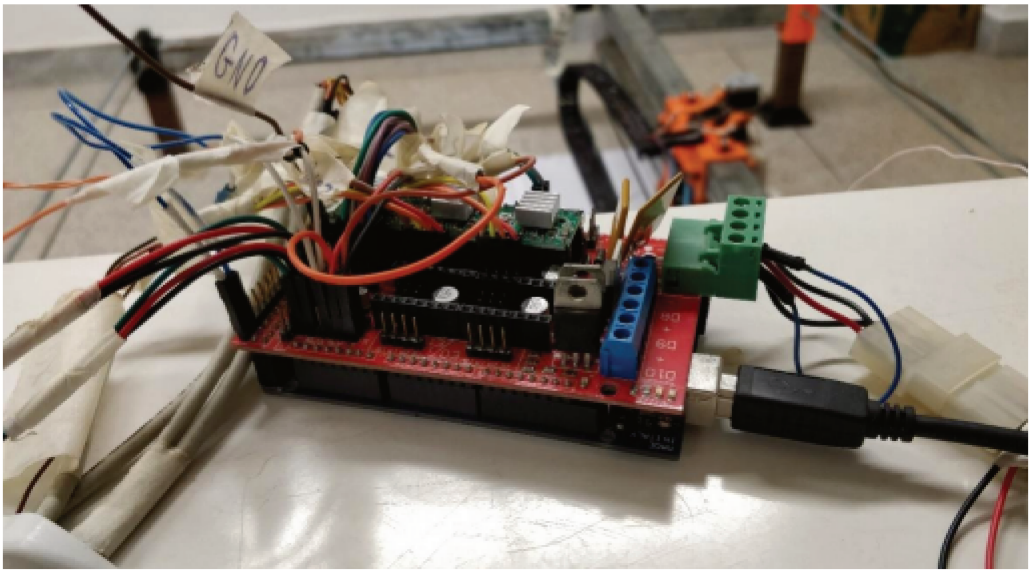

Figura 34: Conjunto Ramps 1.4/Arduino Mega

Fonte: [**29**]

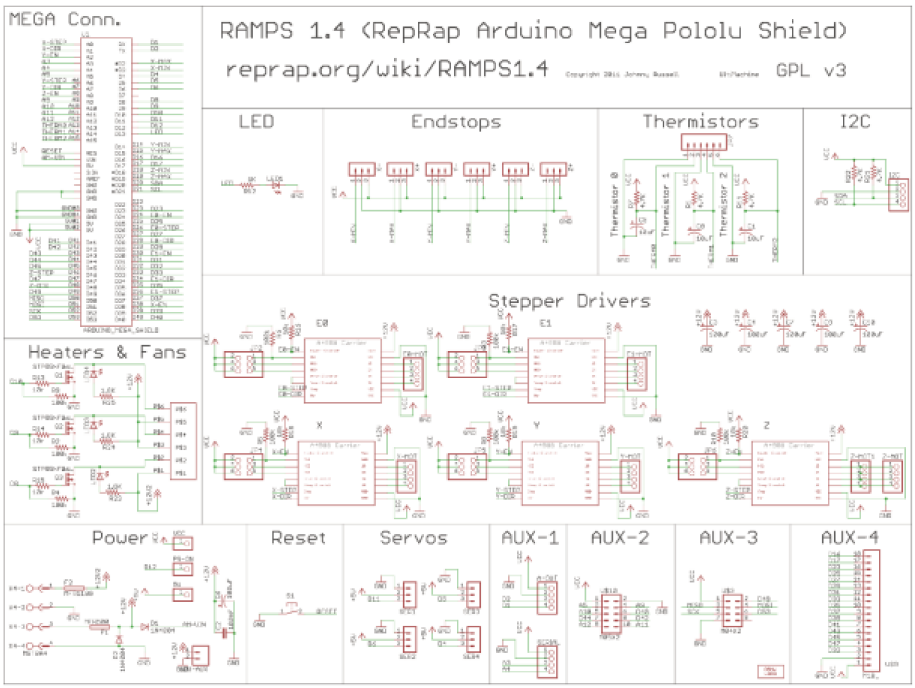

Figura 35: Conexões detalhadas do conjunto

Fonte: <https://user-images.githubusercontent.com/19560798/41185260-bef28948-6b86-11e8-9929-f303c593b4ae.png> (Acesso em: 23/02/2024)

5.5 Sistema Físico completo

Após especificar as características para o funcionamento do robô é possível listar todos os materiais físicos necessários para o funcionamento do projeto.

- Placa Arduino Mega 2560
- Notebook para utilização da IDE do Arduino
- Placa Ramps 1.4
- Fonte de alimentação ATX
- Drivers A4988 para os motores de passo
- Fios e jumpers para conexão
- Motores de passo Nema 17
- Sensores fim de curso
- Leitor RFID PN532

O projeto tem a finalidade de integrar estes elementos fisicos com os softwares desenvolvidos para a PRD.

5.6 Software desenvolvido para a PRD

O controle adaptativo proposto neste trabalho define a execução do Planejamento automático utilizando dados da PRD com a adaptação em situações em que existirem exceções tratáveis. As exceções tratáveis neste trabalho indicam a possibilidade de erros em escrita de informações nas tags, possibilitando então a geração de *snapshots* com defeito. Como já demonstrado nas seções 5.1 a 5.3, existem exceções diretamente ligadas ao módulo de Percepção, e uma ligada ao módulo de execução.

O Software então tem os seguintes objetivos:

- Criação de arquivo *domain.pddl* que definirá todas as condições do domínio utilizado;
- Definir a possibilidade de retorno a posição de referência inicial a cada reinicio do sistema;
- Garantir uma varredura sistemática que tem por objetivo procurar a identificação de ID de cada bloco, definir a posição segundo a Figura 31. Além de se movimentar considerando a simulação de blocos empilhados em uma mesa;
- Parar a varredura ao ocorrer o fim das movimentações ou garantir a identificação de no máximo 5 blocos.
- Conectar o sistema com um programa em Python, utilizando o pacote Pyserial [**3**];
- Definir uma classe para a conexão de variadas placas se necessário, além de métodos para a conexão, captura de caracteres da comunicação Serial, envio de informações e atualização de placas utilizadas;
- Obtenção de caracteres da conexão Serial, a partir de um tempo específico e armazenamento destes caracteres em um arquivo de texto.

- Criação da função **CheckException** em Python que deve filtrar as informações deste arquivo de texto,recolhendo mapa de situação atual, posições de cada bloco em relação a cada coluna, armazenamento de informações da tag correspondentes a cada bloco,obtendo assim predicados, e com estas informações fazer a checagem de exceções que se caso confirmadas retornam um valor;

- Garantir que caso possuam exceções, o programa possa, ao terminar suas checagens, enviar uma informação via comunicação Serial para o programa em Arduino que irá executar todas as mvovimentações necessárias.

- Criação de programa que, caso o retorno desta função garantir um numero, outra função é chamada, cujo nome é **ReWrite** que irá reescrever todas as informações em tags erradas para as informações coerentes com o *snapshot*.

- Ao se tratar as exceções, ou caso o arquivo não possuir nenhuma, realizar a criação da função **PddlEditor** que irá criar ou reescrever o arquivo *problem.pddl* garantindo a conformidade com os predicados estabelecidos no arquivo do domínio além de conformidade com o estado do *snapshot*;

- Criação de função que chama o programa PDDL4J, armazena a saída da execução em outro arquivo de texto, e filtra deste arquivo apenas o plano retirado.

- Obter a sequência de ações por meio de uma lista, que garantirá a indexação de cada comando do plano.

- Resumir as informações que estão na lista, em apenas códigos decifráveis para a movimentação do Arduino.

- Realizar a movimentação conforme o comando e se houver alguma alteração da configuração dos blocos já garantida pelo *snapshot*, determinar uma mensagem de identificação do bloco alterado ou faltante na posição.

- Reiniciar o sistema com uma nova varredura caso esta condição se confirmar, garantindo uma nova geração de *snapshot*.

- Caso contrário, continuar com o envio de comandos do plano.

Os passos definidos anteriormente, garantem a adaptabilidade do robô sem a necessidade de interação humana fazendo assim que o esquema da Figura 20 seja implementado. As Figuras 37 e 38 demonstram a cada módulo de tratamento de Exceções, tanto Estáticas quanto Dinâmicas desenvolvidas estas em Python.A Figura 36 define todo funcionamento do software que foi implementado no robô linear e este define de forma prática todas as etapas de desenvolvimento passando por Percepção, Planejamento e Execução.

A Figura 37 demonstra o processo para se obter a informação correta do posicionamento de cada bloco, e ao se obter algum erro fora do programado, uma mensagem de erro deve ser exibida. Ao se obter a resolução destas exceções, o planejador é acionado.

Já a Figura 38 define todas as etapas que compreendem a verificação de exceções estáticas ao se obter problemas como reconfiguração de blocos no momento do processo de planejamento.

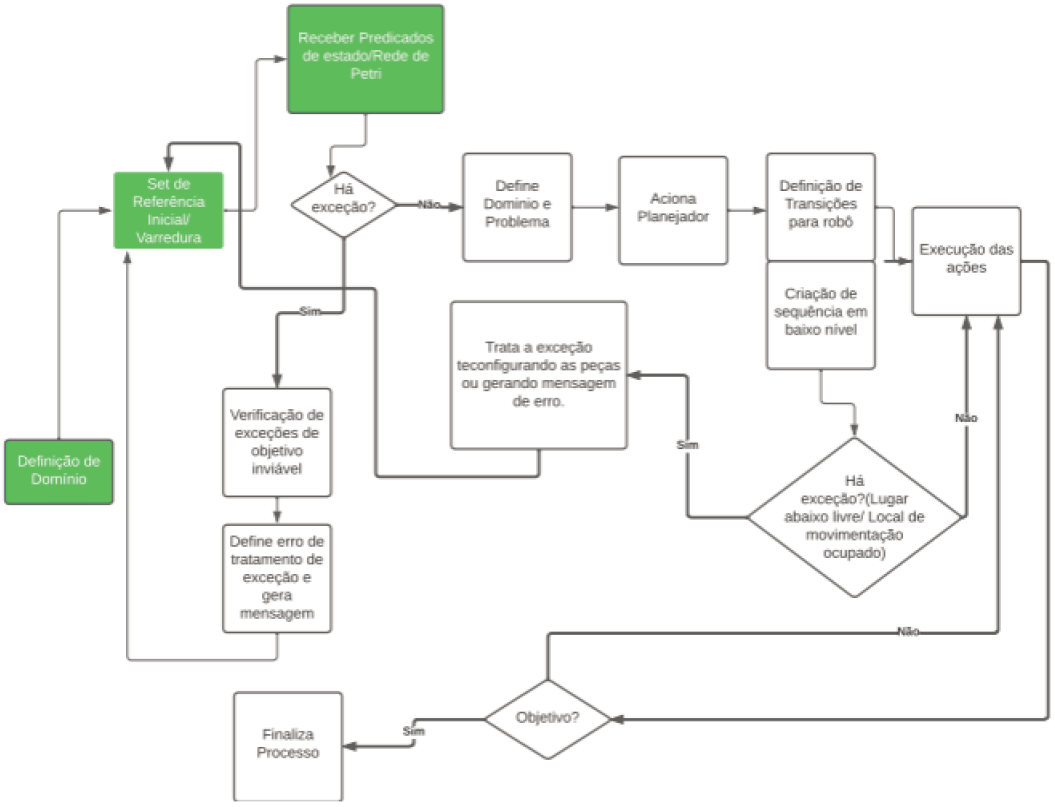

Figura 36: Etapas básicas na execução do Software

Fonte: Fonte: Desenvolvido no site: <https://lucid.app/>(Acesso em: 15/11/2023)

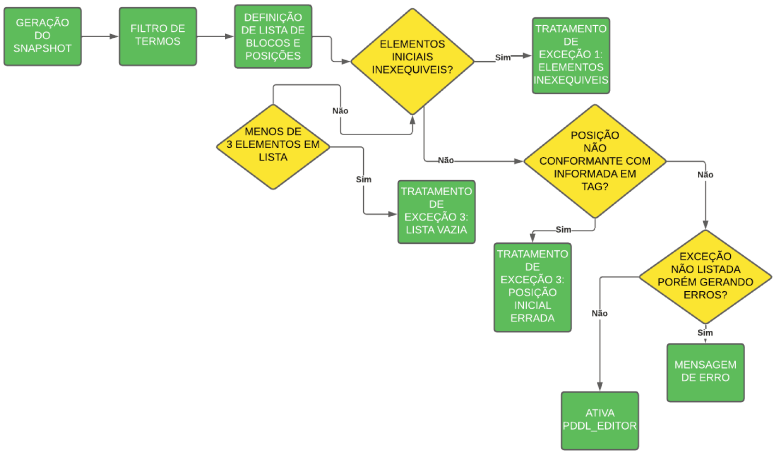

Figura 37: Processo de Checagem de Exceções Estáticas

Fonte: Desenvolvido no site: <https://lucid.app/>(Acesso em: 23/02/2024)

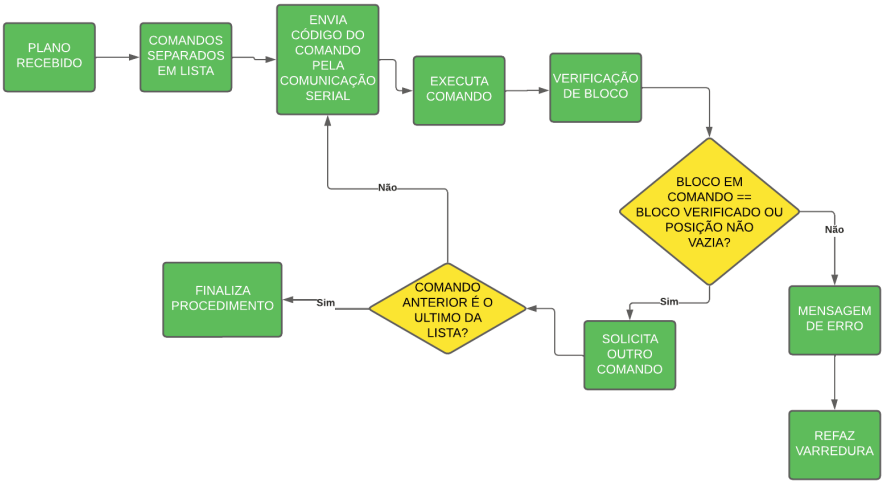

Figura 38: Processo de checagem de Exceção Dinâmica

Fonte: Desenvolvido no site: <https://lucid.app/>(Acesso em: 23/02/2024)

6 RESULTADOS E DISCUSSÕES

A Implementação de todo sistema proposto foi desenvolvida utilizando as etapas práticas seguintes:

- Reutilização de Código para adaptação em mais de 5 blocos
- Desenvolvimento do arquivo *SerialApp.py* para conexão ao Arduino
- Definição de Domínio
- Desenvolvimento das funções do arquivo *CheckException.py*
- Desenvolvimento da função *ReWrite*
- Desenvolvimento da função *PddlEditor*
- Utilização do planejador PDDL4J
- Testes de Checagem de Exceções Estáticas
- Teste de Checagem de Exceção Dinâmica

Cada um destes passos será mostrado nas seções seguintes.

6.1 Adaptação de Codigo

O funcionamento do robô linear utilizando processos de Inteligência Artificial já tinha sido desenvolvido por [**29**] no final do ano de 2023. Este código compreendia funções de arduino que implementavam o modelo da PNRD e iPNRD[**29**] e além disso movimentam o robô em um espaço de trabalho de 60 cm x 60 cm. Porém, este modelo apenas trabalha com 3 blocos, e a PRD pode trabalhar com mais. Então o código foi adaptado para que na varredura, o sistema tivesse um limite de 3 a 5 blocos.Este código se apresenta no site GitHub com todas as suas versões definidas. Com este codigo estabelecido no GitHub o projeto será aberto para outros desenvolvedores.[1]

Além desta adaptação, outras foram feitas neste código:

[1]O código pode ser encontrado no link:
https://github.com/profissionalGuilhermeVitor/SISTEMA_PRD

- Criação de vetor de Strings contendo posições associadas aos simbolos númericos de piso e blocos.
- Adaptação ao texto mostrado no Serial
- Adição de funções de Escrita, leitura e exibição do texto armazenado no *Payload*

6.2 Desenvolvimento de conexão com arduino

O pacote **PySerial** contem métodos e funções que garantem a conexão com a porta USB por meio da comunicação COM, utilizada no Arduino. O *baudrate* utilizado para este projeto foi de 9600 e a função **ReceiveData** foi capar de traduzir e exibir por meio da função **print**, todas as atualizações presentes na comunicação Serial. A Figura 39 exibe parte do código que faz a comunicação e a Figura 40 exibe a função **ReceiveData**.

```
serialP = SerialApp()
serialP.serialPort.port = 'COM5'
print(serialP.updatePort()[0])
```

Figura 39: Conexão da porta COM5 por meio da instanciação de objetos

Fonte: Acervo Pessoal

A Figura 39 demonstra a conexão do Arduino por meio da instanciação do objeto **serialP** na classe **SerialApp**, isto cria a definição, utilizando o paradigma de orientação a objetos, de que o Arduíno Mega utilizado possui os métodos definidos na classe definida.

```
#Receber Dados
def receiveData(self):
    dataRead = ""
    dataRead = self.serialPort.read().decode("latin-1")

    return dataRead
```

Figura 40: Função **receiveData**

Fonte: Acervo Pessoal

Para que seja possível enviar dados para a comunicação Serial a função **sendData** foi criada. Esta função define que a informação passada como parâmetro deve ser enviada por meio de bits que por meio da função **encode**, será convertida em dados necessários para o arduíno. Para fechar a comunicação, a função **closePort** foi criada.A Figura 41 mostra estas duas funções.

```python
#Enviar Dados
def sendData(self,data):
    if(self.serialPort.isOpen()):
        dataSend = str(data) + '\n'
        print("Mensagem " +dataSend)
        self.serialPort.write(dataSend.encode())
        self.serialPort.flushOutput()
        return dataSend

#Fechar a porta

def closePort(self):
    self.serialPort.close()
```

Figura 41: Função **sendData** e função **closePort**

6.3 Definição do Domínio da PRD

Conforme apresentado na Figura 42 o domínio então estará estabelecido como possuindo 2 ações: *pickBlock* e *stack* onde a primeira possui ações de se movimentar até e o bloco escolhido e agarrá-lo com a garra, enquanto a segunda movimenta até a posição escolhida e solta o bloco. Este domínio representa as duas ações com precondições e efeitos definindo quando para o planejador qual sera a consequência de cada ação efetuada. A Figura 13 demonstra o plano com suas ações, sendo que todas elas possuem custo unitário, definindo que ao final, o plano possui um custo total de todas as ações somadas.

```
(define (domain blocks-domain)
(:requirements :strips :typing :disjunctive-
Show hierarchy
(:types
    block - object
    gar - object
)

(:predicates
    (inhand ?block - block)
    (emptyhand)
    (on ?block ?on_block - block)
    (clear ?block - block)
    (table ?block - block)
)
```

```
(:action stack
    :parameters
    (
        ?b - block
        ?t - block
    )
    :precondition
        (and
            (inhand ?b)
            (not(emptyhand))
            (or
                (and
                    (table ?t)
                    (clear ?t)
                )
                (clear ?t)
            )
            (not(= ?b ?t))
        )
    :effect
        (and
            (on ?b ?t)
            (not(clear ?t))
            (clear ?b)
            (not(inhand ?b))
            (emptyhand)
        )
)
```

```
(:action pickBlock
    :parameters
    (
        ?b - block
        ?t - block
    )
    :precondition
    (and
        (not (inhand ?b))
        (not (inhand ?t))
        (not(table ?b))
        (on ?b ?t)
        (emptyhand)
        (clear ?b)
        (not(= ?b ?t))
    )
    :effect
    (and
        (inhand ?b)
        (not(emptyhand))
        (clear ?t)
```

Figura 42: Domínio do Sistema

Fonte: Arquivo Pessoal

Nesta figura se definem os tipos, predicados e ações. Os tipos são os objetos do bloco e da garra, que serão uteis para o planejamento. Em relação aos predicados temos cada um especificado abaixo:

- **(inhand ?block - block)** - Pergunta se o bloco está sendo agarrado no momento.
- **(emptyhand)** - Pergunta se a garra está vazia.

- **(on ?block ?on_block - block)** - Predicado de posicionamento que define qual objeto que está acima e qual está abaixo. Por exemplo, se tivermos B acima de A, a representação verdadeira do predicado seria **(on B A** e se caso tivéssemos o bloco C no piso da coluna 1 teríamos **on C t1** como predicado verdadeiro.

- **(clear ?block - block)** - Define se objeto está livre de blocos acima dele

- **(table ?block - block)** - Define se objeto é a figura da mesa, e estes elementos definem qual coluna estamos verificando, por exemplo, **"t4"** significa que estamos observando a quarta coluna do PSS.

6.4 Desenvolvimento das funções do arquivo *CheckException.py*

Para definir o problema utilizando o sistema obtido nesta implementação, alguns passos devem ser alcançados. Primeiramente, temos que definir um modelo de problema onde podemos ter um repositório de objetos, um *snapshot* padrão e os objetivos em um arquivo de problema. Para obter este arquivo escrito em extensão PDDL, um módulo em Python precisa se comunicar com o módulo do controlador, que nesta implementação está embarcado em uma placa Arduino.

Publicando os dados por meio da comunicação serial do Arduino, o comando pyserial envia todos os caracteres definidos, possibilitando armazená-los em um arquivo de texto,definido como o *snapshot*, como é possível observar na Figura 43.

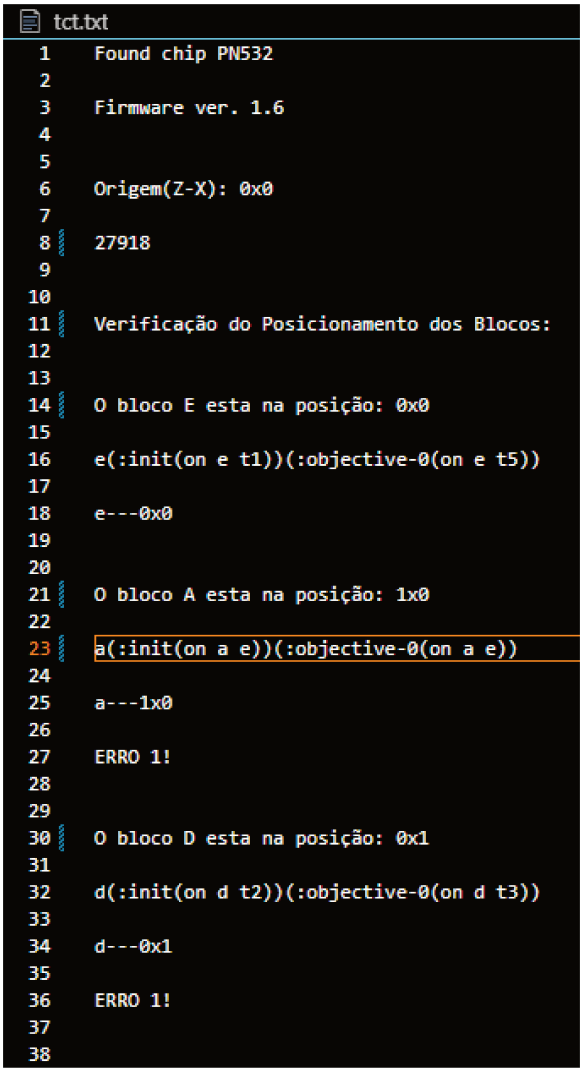

Figura 43: Snapshot definido em arquivo de texto

Fonte: Acervo pessoal

Este arquivo mostra a posição real de cada bloco além das informações presentes na tag. A identificação do bloco é feita por caracteres de 'a' até 'e', sendo estes representados com letras minúsculas e logo após dois predicados: ':init' e ':objective-0', contendo em cada um deles um predicado de posicionamento ('on bA bB'), que neste exemplo possui o significado de: "bloco A está acima do Bloco B". Quando "bB"é identificado como mesa ("t1"a "t5"), o significado muda para: "bloco A acima da mesa tN". A verificação de estado é obtida por meio de uma varredura em todo o espaço de trabalho identificado na Figura 10 e possui algumas restrições:

- É preciso encontrar pelo menos 3 blocos;
- A varredura finaliza ao se encontrar 5 blocos;

- Ao encontrar um bloco em alguma posição, o elemento ativo adiciona 6 unidades em sua movimentação, variando assim sua posição no eixo x;
- Ao não encontrar um bloco na posição nxn, o monitor serial emite uma mensagem de "ERRO 1"e adiciona uma unidade, variando sua posição no eixo y.

Com a geração do arquivo de varredura, então se verifica a possibilidade de exceções. Nesta parte do processo, procuram-se algumas exceções específicas[2]:

- Exceção 1i: Blocos em posição inicial diferente do PSS;
- Exceção 1o: Blocos com objetivos finais conflitantes;
- Exceção 2: Blocos com informações objetivo inequívocas;
- Exceção 3: Informação vazia de posição inicial;
- Exceção 4: Informação vazia de posição objetivo.

Como posição teste para a implementação do sistema, utilizam-se os blocos configurados como mostrado na Figura 44 (bloco "a"sobre bloco "e", bloco "e"sobre a mesa "t1", bloco "d"sobre a mesa "t2", bloco "c"sobre a mesa "t3"e bloco "b"sobre a mesa "t5") e sua posição objetivo como apresentado na Figura 45 (bloco "c"sobre o bloco "b", bloco "b"sobre o bloco "d", bloco "d"sobre a mesa "t3", bloco "a"sobre o bloco "e", bloco "e"sobre a mesa "t5").

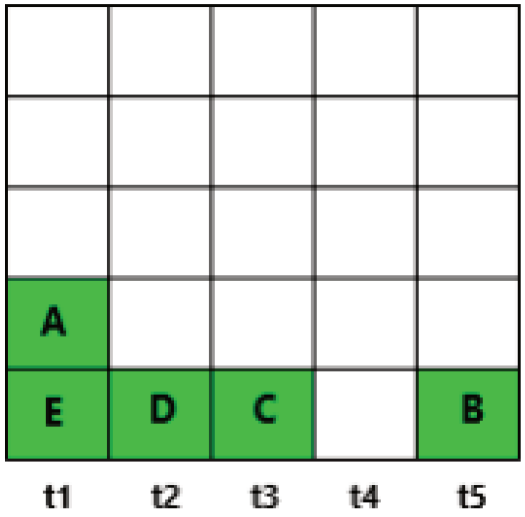

Figura 44: Posição Inicial de teste

Fonte: Acervo pessoal

Para que o tratamento da informação possa ser feito, algumas exceções listadas, apenas podem apresentar uma mensagem de Erro pois o sistema implementado não conduz a passagem de parametros de usuário. Então as exceções 1o, 2 e 4 ainda não podem ser tratadas pois consideram o tratamento de informações dos predicados *:objective-0*.

Para tratar as informações e identificar as exceções 1i e 3 é necessário obter os carácteres escritos no *snapshot* e filtrar as informações correspondentes ao posicionamento, blocos, posições originais e seus equivalentes para assim estabelecer a lógica de identificação de cada uma.

[2]O vídeo de demonstração de todas as exceções possíveis se encontra em: https://www.youtube.com/watch?v=MttXGEPcY8s

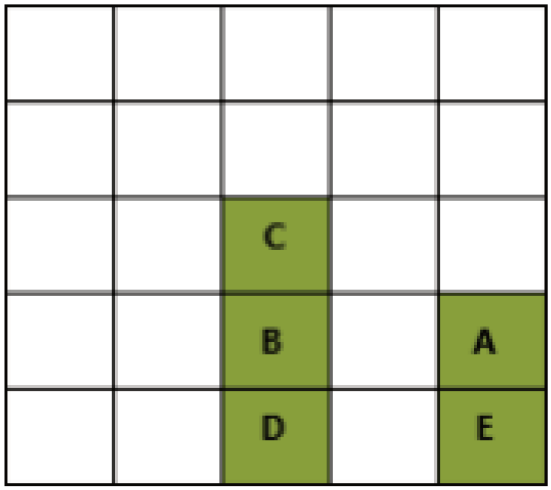

Figura 45: Posição Objetivo de teste

Fonte: Acervo pessoal

A Exceção 1i[3] considera que a posição dos blocos está errada em relação à posição da tag. Na Figura 46 podemos ver que o predicado com a marcação *:init* esta indicando que o bloco A está na posição "t1", que seria a mesma posição que o bloco E que também estaria na posição "t1".Porém, como é possível perceber na Figura 47 o bloco A está acima do Bloco E. Para então resolver este problema é necessário haver a identificação no *snapshot*, de posições repetidas. Para executar esta filtragem se utiliza o modulo **re[27]** que possui funções e métodos de Expressões Regulares(ReGex). As expressões regulares identificam padrões no predicado *:init* que garante o armazenamento de uma lista com todas as posições no predicado citado, assim ativando a exceção e armazenando os blocos envolvidos nesta. A função de identificação envia uma mensagem contendo o tipo da exceção e envia diretamente para o programa principal. Para se tratar esta exceção, é necessária esta função de identificação enviar a lista contendo os blocos envolvidos e ao chamar a função **writeNewInfo** que se apresenta no arquivo **ReWrite.py**.

```
O bloco E esta na posi��o: 0x0

e(:init(on e t1))(:objective-0(on e t5))


O bloco A esta na posi��o: 1x0

a(:init(on a t1))(:objective-0(on a e))

ERRO 1!
```

Figura 46: Informação nas tags

Fonte: Acervo pessoal

A Exceção 3 é definida ao não haver informação presente no predicado*:init*, seja com informação vazia entre parentesis, seja com nenhuma informação com esta marcação, o exemplo de informação em tag se apresenta na Figura 48[4] .

[3]O vídeo de demonstração do tratamento desta exceção se encontra em: https://www.youtube.com/watch?v=166H7d2IkFc

[4]O vídeo de demonstração do tratamento desta exceção se encontra em: https://www.youtube.com/watch?v=6XmY51XPDas

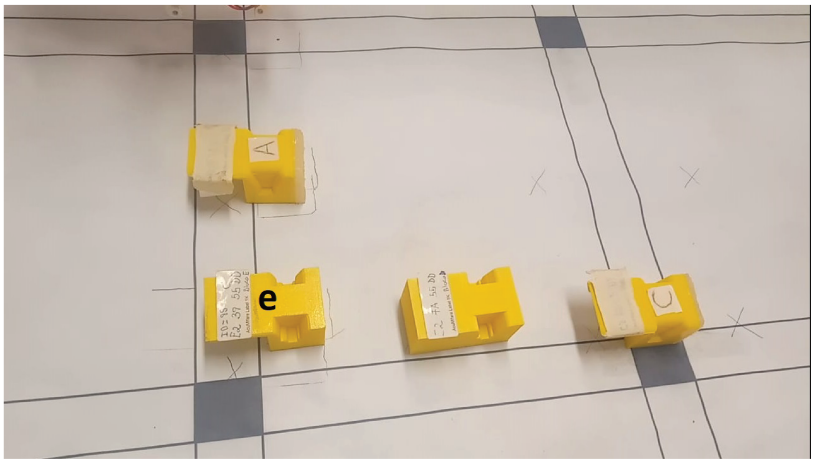

Figura 47: Posicionamento Real

Fonte: Acervo pessoal

```
O bloco A esta na posição: 0x0
(:init())(:objective-0(on a t1))

O bloco B esta na posição: 1x0
(:init(on b a))(:objective-0(on b a))

O bloco C esta na posição: 2x0
(:init(on c b))(:objective-0(on c t2))

O bloco D esta na posição: 3x0
(:init())(:objective-0(on d t4))

O bloco E esta na posição: 4x0
(:init(on e d))(:objective-0(on e t3))
```

Figura 48: Exceção 3 nos blocos A e D

Fonte: Acervo pessoal

6.5 Criação do arquivo *ReWrite.py*

A função *writeNewInfo* recebe um código de exceção e as informações presentes do retorno das funções em **CheckException**. Estas informações tendem a ser o código da exceção e os blocos envolvidos armazenados em uma lista. Assim, com associações de posicionamento feitas por RegEx, onde há a atualização do posicionamento dos blocos e identificação do predicado necessário temos a reescrita da tag seguindo o seguinte modelo:
[letra correspondente do bloco](:init(on b1 b2))(:objective-0(on b1 b3)).

A Figura 49 demonstra este resultado na simulação em Python.

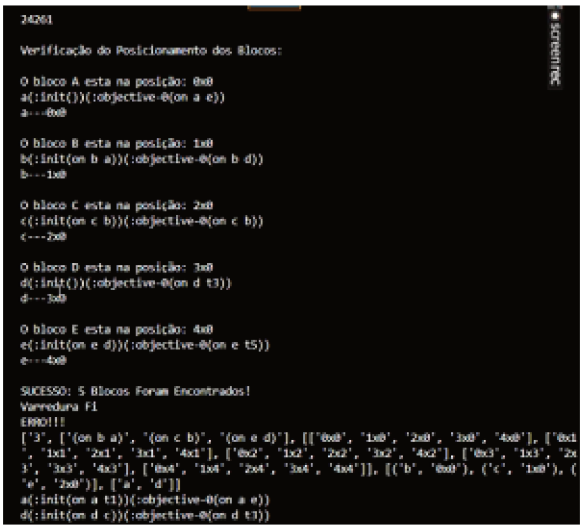

Figura 49: Reescrita de Arquivos e identificação de novas informações

Fonte: Acervo pessoal

6.6 Desenvolvimento da Função *PddlEditor*

Esta função é necessária após haver o tratamento de exceções no programa principal. Ela tem o objetivo de reescrever um arquivo *problema.pddl* utilizando de informações necessarias apenas os predicados recebidos no *snapshot*. O arquivo de problema obtido é demonstrado na Figura 50 e ele apresenta outros predicados já definidos no domínio da Figura 42, como por exemplo o predicado **(clear bloco)**. Por meio das funções escritas neste programa é possível obter um mapa de posicionamento do *snapshot* que define quais blocos estão livres ou não e os escreve em predicados.

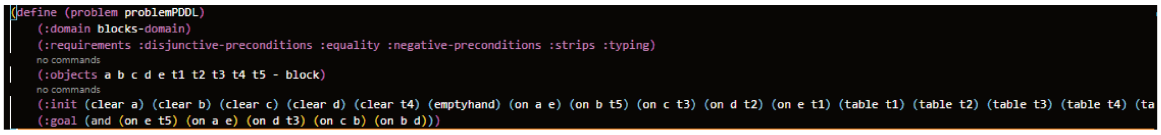

Figura 50: Arquivo de problema

Fonte: Acervo Pessoal

Além destes predicados contém predicados de definição de mesa, que já são definidos por padrao, e há a inserção dos predicados de objetivo neste arquivo, deixando assim a possibilidade de se aplicar um planejador automático utilizando os arquivos *domain.pddl* e *problem.pddl*. Como mostrados em pasta na Figura 51.

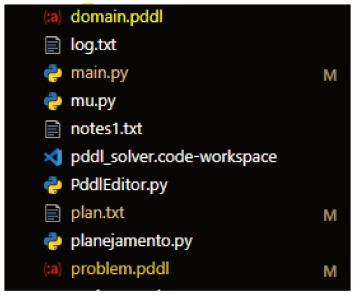

Figura 51: Arquivos de Domínio e Problema

Fonte: Acervo Pessoal

6.7 Acionamento do Planejador PDDL4J

Ao se reeditar o arquivo de problema, o planejador PDDL4J integrado no IDE do VS Code aplica o processo de planejamento e retorna como saída, valores de custo, tempo de processamento e comandos diretos do seu plano. Para conseguir estes caracteres é necessário se utilzar o módulo Python **Subprocess** que possui funções que retornam valores da saída de comandos de Terminal.

O plano possui dois comandos: **pickBlock** e **stack**. O primeiro comando executa a movimentação do robõ até a posição do bloco e a identificação do próprio. Este comando possui a seguinte codificação:**pickBlock (bloco que será manipulado) (posição ou bloco abaixo)**

Obtendo a identificação deste bloco e sabendo que o código de Arduino armazena a posição deste é possível haver a movimentação do braço robótico para a posição específica do bloco.

O comando **stack** executa o movimento de retirar o bloco da sua posição e inseri-lo em outra posição. E este comando possui a seguinte codificação:**stack (bloco retirado) (posição ou bloco acima)**. Ao se identificar a posição do bloco, o robô o posiciona.

Para que haja a comunicação do Arduino com este planejador, os comandos terão uma codificação mais simplificada:
[primeira letra do comando][b1][b2]

O exemplo da Figura 52 apresenta o planejador exibindo o plano completo e a Figura 53 apresenta estes comandos simplificados.

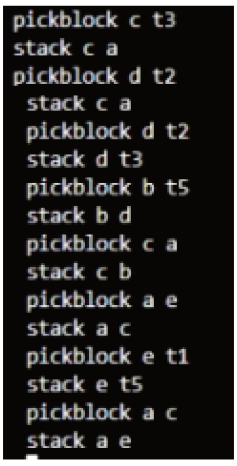

Figura 52: Plano do Sistema

Fonte:Acervo Pessoal

```
pickblock d t2
stack d t3
pickblock b t5
stack b d
pickblock c a
stack c b
pickblock a e
stack a c
pickblock e t1
stack e t5
pickblock a c
stack a e
['pct3', 'sca', 'pdt2', 'sdt3', 'pbt5', 'sbd', 'pca', 'scb', 'pae', 'sac', 'pet1', 'set5', 'pac', 'sae']
```

Figura 53: Comandos do Planejamento Codificados

Fonte:Acervo Pessoal

6.8 Testes de Checagem de Exceções Estáticas

Quando há alguma exceção estática no sistema, uma função de tratamento é chamada, avaliando o arquivo de *snapshot* chamado de *snapshot.txt*. Este arquivo possui a extensão *.txt* e apresenta sequencialmente a exibição em texto do sistema em Arduíno. Na Figura 54 tem-se o texto de reconhecimento do chip do leitor RFID PN532, após este reconhecimento, temos a movimentação do robô ao seu ponto de origem 0x0.

A varredura é acionada com o robô verificando os blocos até encontrar o final de seu espaço de trabalho ou haver a verificação de no máximo 5 blocos. A Figura 54 mostra o robô em execução junto com a exibição automática de seus dados.

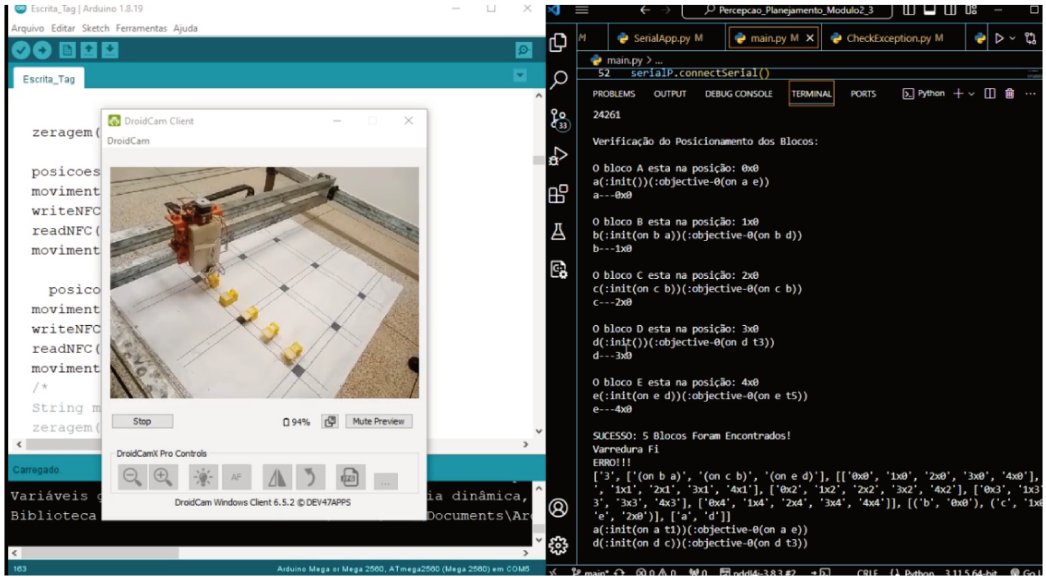

Figura 54: Captura em tempo real do robô em funcionamento

A transmissão destes dados finaliza e o programa principal recebe o resultado das funções no arquivo **CheckException.py**. Ao serem retornados valores de texto segundo os tipos das exceções, o tratamento de exceções conforme a funcão **writeNewInfo** é chamado, encontrando a base de informações e blocos a se reescrever como demonstrado na Figura 49.

Assim, uma nova varredura é feita e um novo *snapshot* é encontrado[5].

Os testes feitos presentes na Figura compreendem um posicionamento onde todos os blocos estão na mesa **"t1"** e as informações dos blocos "a"e "d"estão erradas. O acionamento deste tratamento e visto na linha de código da Figura 55. Nesta linha de código, o objeto **exEstatica** que compreende a uma instanciação de **CheckExcep-**

[5]Os vídeos de demonstração do tratamento de exceções estão presentes nos links:
https://www.youtube.com/watch?v=166H7d2IkFc
https://www.youtube.com/watch?v=6XmY51XPDas

tion procura exceções e as trata até o retorno ser diferente de **None** o que significa que não teria nenhuma exceção.Ao encontrar este resultado o programa prossegue.

```
while (exEstatica.exception1() is not None):

    print(exEstatica.exception1())
    time.sleep(10)
    print(ReWrite.writeNewInfo(exEstatica.exception1()))
    for c in ReWrite.writeNewInfo(exEstatica.exception1()):
        print(c)
        serialP.serialPort.write(c.encode())
        time.sleep(40)
    serialP.closePort()

    serialP.connectSerial()
    with open('snapshot.txt','w') as writer:
        writer.write(c)
```

Figura 55: Código de Busca e tratamento de exceções

6.9 Teste de Checagem da Exceção Dinâmica

Não havendo nenhuma exceção no andamento do programa, uma função no programa principal como mostrado na Figura 56, irá executar o PDDL4J por meio do código em terminal demonstrado pela Figura 57. Então ao terminar este processo, o plano é exibido e irá retornar uma lista com os códigos resumidos.

```
def planejamento(e):
    try:
        editor = PddlEditor()
        editor.editar_problema(e)
        actionsList = plan()
    except:
        print("PLANO NÃO PODE SER REALIZADO")
    for c in actionsList:
        time.sleep(15)
        serialP.serialPort.write(c.encode())
        time.sleep(15)
        d=""
        ti = timeit.default_timer()
        tf = timeit.default_timer()
        while tf-ti<15.7:
            tf = timeit.default_timer()

            d += serialP.receiveData()
        with open('testando.txt','w') as writer:
            writer.write(d)
        if("ERRO!!!! BLOCO NAO IDENTIFICADO-REALIZAR NOVA VARREDURA!!" in d):
            print(f"ERRO DE PLANEJAMENTO-NOVA VARREDURA SERÁ REALIZADA- BLOCO {c[1]} NÃO IDENTIFICADO")
            return False
        else:
            continue
```

Figura 56: Código em Python do planejamento

```
                                                                    > java -jar C:\Users\Home\Desktop\teste
\pddl4j\pddl4j-3.8.3.jar -o C:\Users\Home\GitHub_Files\SISTEMAS_PRD_PNRD_iPNRD\Programa_PRD\PRD\Percepcao_Planejamento_Modulo2_3\domain.pddl -
f C:\Users\Home\GitHub_Files\SISTEMAS_PRD_PNRD_iPNRD\Programa_PRD\PRD\Percepcao_Planejamento_Modulo2_3\TestesExemplo\problem.pddl -p 0
```

Figura 57: Código de execução do planejador automático

A Figura 56 demonstra que ao instanciar a variável **editor** um método ira editar o problema de acordo com o *snapshot* e executar o planejador utilizando a função **plan()**, esta função gera uma lista de *strings* que terão os codigos de comando como mostrado na Figura 53. Esta Função gera o código de Terminal demonstrado na Figura 57.Este codigo irá enviar por comunicação Serial os codigos em sequência e executar os comandos no módulo robótico.

O primeiro comando da Figura é o **pct3** que significa que o robô irá executar o comando **pickBlock**, irá agarrar o bloco **c** que está localizado em **"t3"**[6].

A Exceção Dinâmica pode ser checada ao se retirar o bloco do primeiro comando do plano da posição que este se encontra, fazendo com que o robô tenha que fazer uma nova varredura e re-gerar o arquivo de *snapshot*. A Figura 58 mostra o robô definindo o plano com o bloco e executando o comando com o estado alterado[7].

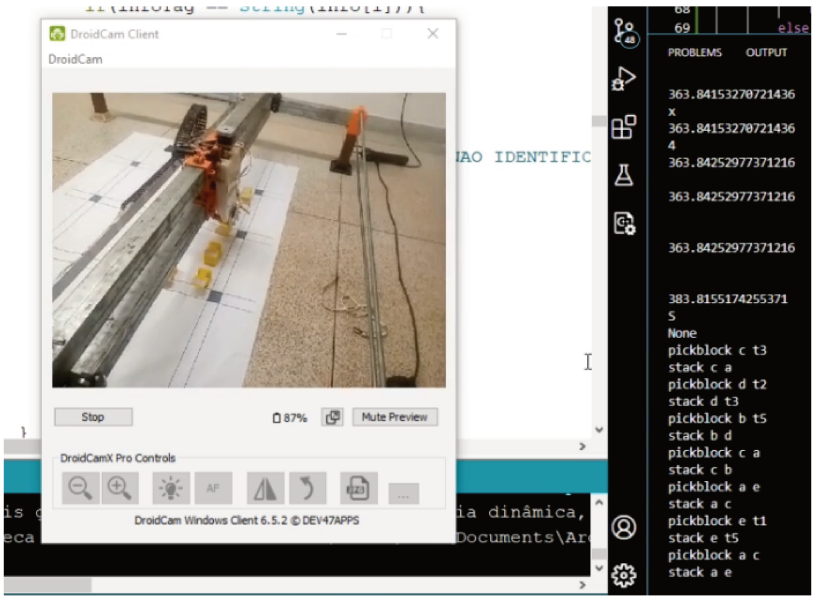

Figura 58: Fim da geração do Plano

A Figura 60 demonstra o estado alterado, com o bloco "c"sendo retirado e a Figura 59 demonstra a mensagem exibida assim que o robô faz a verificação para a ação *pickBlock*. No programa de Arduino, assim que o bloco é identificado, sua posição real é armazenada, e ao consultar este bloco, a ação irá exatamente na posição definida, na varredura.

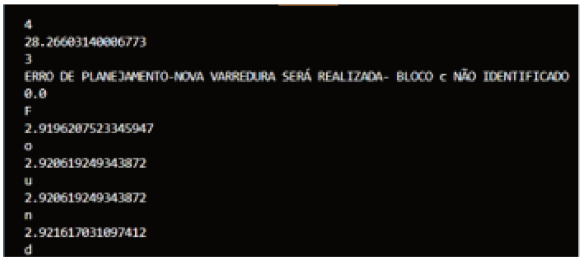

Figura 59: Mensagem de Exceção Dinâmica

[6]Lembrando que ao aparecer uma mesa, significa que o robô está localizado na posição dita, porém se o que aparecer no código for um bloco, significa que o bloco selecionado estará acima deste.

[7]O video de demonstração da exceção dinâmica se encontra em: https://www.youtube.com/watch?v=6XmY51XPDas

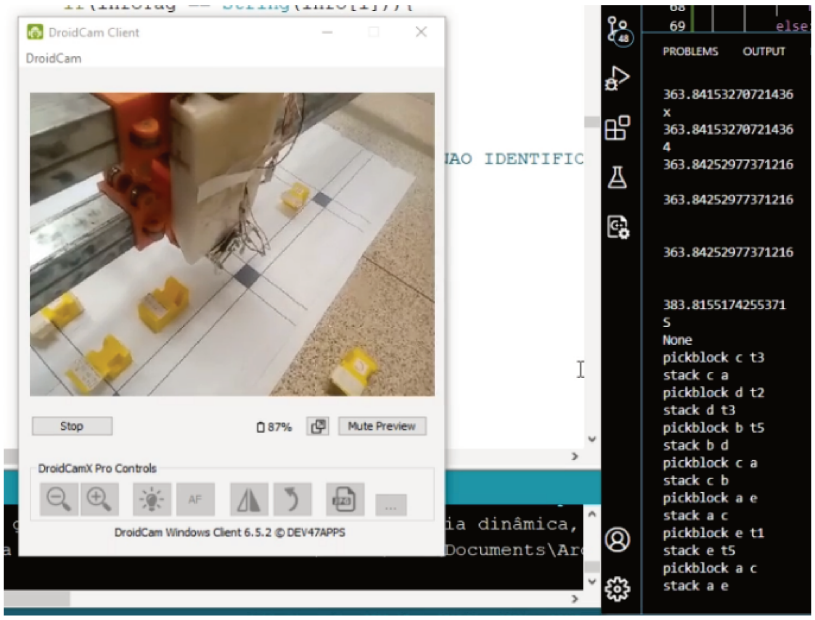

Figura 60: Checagem da Exceção Dinâmica

7 CONCLUSÃO

O trabalho proposto anteriormente por [**32**] e [**33**] define a implementação do modelo em um robô linear, e como visto neste trabalho o objetivo foi alcançado por demonstrar o tratamento de exceções e da resolução de um problema de planejamento integrado à PRD. O projeto teve exito em seu funcionamento, e a comunicação com a porta Serial foi satisfeita. O funcionamento deste trabalho demonstrou a eficácia do uso da PRD em uma aplicação que utiliza a conexão física e a integração de elementos como principal recurso.

O domínio apresentado no teste compreende uma aplicação para a resolução do problema do Mundo de Blocos, mas o sistema construído pode ser aplicado em diversos domínios apenas com a mudança do arquivo de domínio integrado com o módulo PddLEditor. O sistema, também contendo modularidade, pode ser integrado por meio de multi-agentes, reduzindo assim a presença humana.

Este trabalho contribui com a implementação direta de um sistema utilizando planejadores automáticos aplicados em robô didático, o que indica que sua utilização pode ser feita diretamente em aplicações industriais. Essa abordagem abre espaço para robôs mais autônomos e capazes de resolver exceções aos processos, todavia necessita de uma modelagem de domínio adequada.

8 TRABALHOS FUTUROS

Limitações podem ser encontradas em relação à memória do leitor RFID, devido ao seu tamanho limitado, além da distância de leitura e escrita de cada leitor utilizado. A arquitetura do controlador, além da linguagem de comandos em baixo nível, pode ser alterada para outras que possuem suporte à linguagem Python, deixando a integração mais otimizada e, além disso, contendo um número menor de módulos.

A adaptação do código inicial estabelecido pode ser feita por meio de um aplicativo que tenha uma experiência de usuário mais interativa, através da construção de um ambiente mais gráfico para a melhor compreensão da reescrita das tags e uma melhor visão do plano utilizado, diferente do que se tem utilizando o editor VSCode.

Uma implementação deste sistema em serviços *online* como soluções Cloud/Edge pode ser feita em trabalhos futuros. Além das exceções listadas neste trabalho, tanto as de configuração quanto as de planejamento possuem especificidades em seu tratamento e em suas mensagens de erro, por isso a supervisão de erros não listados e os procedimentos corretos devem ser feitos ao utilizar a estrutura em ambiente industrial, pois o sistema lida com observação parcial, sempre retornando para o ponto inicial.

O padrão de análise de texto pode ser reaproveitado para outros sistemas que utilizam planejamento automático em seus sistemas completos, pois utiliza pacotes de RegEx para definir e filtrar apenas os textos necessários e obter relações entre posições abstratas e reais. Este padrão pode ser melhorado com o uso de inteligências artificiais generativas para uma otimização na busca e relacionamento das strings.

9 REFERÊNCIAS BIBLIOGRAFICAS

Referências

[1] Asada, H., Slotine, J. J. E. (1986). Robot Analysis and Control.

[2] ASSUNÇÃO, Pedro Darc da Cruz. A Matemática na Produção de Jogos Digitais com Inteligência Artificial. 58f. Trabalho de Conclusão de Curso (Licenciatura em Matemática) – Universidade Federal do Tocantins, Araguaína, 2022.

[3] Chris Liechti. PySerial Documentation. Disponível em: https://pythonhosted.org/pyserial/. Acesso em: fevereiro de 2024.

[4] Craig, J. J. (2005). Introduction to Robotics: Mechanics and Control.

[5] da Silva Fonseca, J.P., de Sousa, A.R., Ferreira, M.V.M., Tavares, J.J.P.Z.S. (2016). Planpas: Plc and automated planning integration. International Journal of Computer Integrated Manufacturing, 29(11), 1200–1217. doi:https://doi.org/10.1080/0951192X.2015.1067909

[6] DEVELOPERS, F.-D. G. Fast-Downward Official Documentation. [S.l.], 2018.

[7] de SOUSA, Marcelo Henrique. RFID E SUAS APLICAÇÕES — UM ESTUDO DE CASO COM PRATELEIRAS INTELIGENTES.Tese (Mestrado em Engenharia de Teleinformática) — Faculdade de Engenharia de Teleinformática da Universidade Federal do Ceará. Pernambuco, p. 17. 2010.

[8] Favorito M., Fuggitti F., Muise C. pddl. Disponível em: https://github.com/AIPlanning/pddl. Acesso em: fevereiro de 2024.

[9] FELTY, A.; MILLER, D. Specifying theorem provers in a higher-order logic programming language. In: SPRINGER. International Conference on Automated Deduction. [S.l.], 1988. p. 61–80.

[10] FOWLER, M. UML Distilled: A Brief Guido to the Standard Object Modeling Language. 3. ed. [S.l.]: Addison-Wesley, 2003. ISBN 0-321-19368-7.

[11] Friedl, Jeffrey E. F. "Mastering Regular Expressions."O'Reilly Media, 2006.

[12] Ghallab, M., D. Nau, and P. Traverso. 2014. The Actors View of Automated Planning and Acting: A Position Paper. Artificial Intelligence 208: 1–17. doi:https://doi.org/10.1016/j.artint.2013.11.002.

[13] 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. Yale Center for Computational Vision and Control. Tech Report CVC TR DCS TR. October, 1998.

[14] Guérin, J., Thiery, S., Nyiri, E., and Gibaru, O. (2018). Unsupervised robotic sorting: Towards autonomous decision making robots. International Journal of Artificial Intelligence and Applications.

[15] Hofer, L. (2017). Decision-making algorithms for autonomous robots. Ph.D. thesis.

[16] Jan Dolejsi. Planning Domain Description Language Support. Disponível em: https://github.com/jan-dolejsi/vscode-pddl. Acesso em: fevereiro de 2024.

[17] K. Hitomi, "Automation: Its concept and a short history,"Technovation, vol. 14, no. 2, pp. 121-128, 1994.

[18] KUMAR, G. S. et al. Path planning algorithms: A comparative study. In: . [S.l.: s.n.], 2011.

[19] Latombe, J.-C. Robot Motion Planning. Introduction and Overview, The Springer International Series in Engineering and Computer Science, vol. 124, 1991.

[20] Liu, R.; Nageotte, F.; Zanne, P.; de Mathelin, M.; Dresp-Langley, B. Deep Reinforcement Learning for the Control of Robotic Manipulation: A Focussed Mini-Review. Robotics 2021, 10, 22.

[21] MENEGHETTI, A. Optimizing allocation in floor storage systems for the shoe industry by constraint logic programming. In: IEEE. 2009 Ninth International Conference on Intelligent Systems Design and Applications. [S.l.], 2009. p. 467–472.

[22] Nogueira,I.C. Gerenciando a Biblioteca do Amanhã: tecnologias para otimização e agilização do serviços de informação.2014.11 f.Escola de Ciência da Informação da UFMG, 2014.

[23] NUNES, J. d. S. Uma introdução concisa sobre os fundamentos da Lógica. [S.l.: s.n.], 2020. 367 p.

[24] OLIVEIRA, Nina Cervilha. Adaptação de robô cartesiano para máquina de corte a laser. 2020. 70 f. Trabalho de Conclusão de Curso (Graduação em Engenharia Mecatrônica) – Universidade Federal de Uberlândia, Uberlândia, 2020.

[25] PDDL4J Documentation, PDDL4J website, accessado em 14 de Fevereiro de 2024, http://pddl4j.imag.fr/running_planners_from_command_line.htm.

[26] Prasuna, R.G., Potturu, S.R. Deep reinforcement learning in mobile robotics – a concise review. Multimed Tools Appl (2024). https://doi.org/10.1007/s11042-024-18152-9

[27] Python Software Foundation. Re Documentation. Disponível em: https://docs.python.org/3/library/re.htmlhttps://docs.python.org/3/library/re.html. Acesso em: fevereiro de 2024.

[28] RUSSELL, S.; NORVIG, P. Artificial Intelligence: A Modern Approach. 3rd. ed. Upper Saddle River, NJ, USA: Prentice Hall Press, 2010. ISBN 0136042597, 9780136042594.

[29] Santana,T.A. IMPLEMENTAÇÂO DA INTEGRAÇÃO PNRD E iPNRD PARA MUNDO DE BLOCOS. 2023. 58 f. Trabalho de Conclusão de Curso (Graduação em Engenharia Mecatrônica) – Universidade Federal de Uberlândia, Uberlândia, 2023.

[30] Siciliano, B., Khatib, O. (2008). Springer Handbook of Robotics.

[31] SOARES, Alícia.O que é a linguagem de marcação, os seus principais exemplos e como implementá-la? Voitto, 2022. Disponível em: https://www.voitto.com.br/blog/artigo/linguagem-de-marcacao<https://www.voitto.com.br/blog/artigo/linguagem-de-marcacao>. Acessada em 29 nov 2023.

[32] Souza,G.A. INTRODUÇÃO AO CONTROLE ADAPTATIVO A EVENTOS DISCRETOS EM BASE DE DADOS RFID COM PLANEJADOR AUTOMÁTICO APLICADO A SISTEMAS ROBÓTICOS. 2020. 84 f. Trabalho de Conclusão de Curso (Graduação em Engenharia Mecatrônica) – Universidade Federal de Uberlândia, Uberlândia, 2020.

[33] Souza,G.A., Silva,J.R., Tavares,J.J.P.Z.S. Towards Adaptive Discrete Event Control Based on PRD, PSS and Automatic Planner. In: XXIII Congresso Brasileiro de Automática (CBA 2020), 2020, Santa Maria. Anais do XXIII Congresso Brasileiro de Automática (CBA 2020), 2020. doi:https://doi.org/10.48011/asba.v2i1.1448

[34] Tavares, J.J.P.Z.d.S. and Souza, G.d.A. (2019). Pnrd and ipnrd integration assisting adaptive control in block world domain. INPROCEEDINGS of the International Workshop on Petri Nets and Software Engineering 2019, 73–90.

[35] UGRINOVICH, Victor Santos. Adequação de robô cartesiano para trabalhar utilizando PNRD. 2023. 38 f. Trabalho de Conclusão de Curso (Graduação em Engenharia Mecatrônica) – Universidade Federal de Uberlândia, Uberlândia, 2023.

[36] VAQUERO, T. S. ITSIMPLE: Ambiente Integrado de Modelagem e Análise de Domínios de Planejamento Automático. Dissertação (Mestrado) — Master thesis, Polytechnic School of the University of São Paulo, 2007.