UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE ENGENHARIA MECÂNICA
CURSO DE GRADUAÇÃO EM ENGENHARIA MECATRÔNICA

GABRIEL DE ALMEIDA SOUZA

INTRODUÇÃO AO CONTROLE ADAPTATIVO A EVENTOS DISCRETOS EM BASE DE DADOS RFID COM PLANEJADOR AUTOMÁTICO APLICADO A SISTEMAS ROBÓTICOS

UBERLÂNDIA
2020

GABRIEL DE ALMEIDA SOUZA

INTRODUÇÃO AO CONTROLE ADAPTATIVO A EVENTOS DISCRETOS EM BASE DE DADOS RFID COM PLANEJADOR AUTOMÁTICO APLICADO A SISTEMAS ROBÓTICOS

Trabalho de Conclusão de Curso apresentado no curso de graduação em Engenharia Mecatrônica da Universidade Federal de Uberlândia, como parte dos requisitos para a obtenção de título de **BACHAREL EM ENGENHARIA MECATRÔNICA**

Área de concentração: Engenharia Mecatrônica

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

UBERLÂNDIA
2020

Este trabalho é fruto do acúmulo de minhas experiências.
Obrigado pelo aprendizado!

AGRADECIMENTOS

À Sara, por ser uma pessoa incrível e que acredita no meu potencial.

Aos meus pais e meu irmão Renato, pelo suporte e confiança, pela tolerância nos momentos de estresse, e pelos exemplos de valor e ética. À minha família, pelas reuniões cheias de descontração.

Aos meus amigos que estiveram comigo e compartilham de experiências, sentimentos e esperanças. Ao Gustavo e Henrique, pelas risadas e pela parceria durante a graduação e cotidiano.

Aos meus veteranos de trabalho, pelo ganho de experiência como desenvolvedor e maturidade para debater ideias e compreender o ciclo de vida de produtos tecnológicos.

Aos membros do MAPL, em especial Alan, que trilharam esse caminho comigo e cujos estudos foram contribuições fundamentais para este trabalho. Aos membros da EPTA, pela amizade e pelas múltiplas oportunidades de crescimento e experiências únicas. Aos discentes e docentes do curso de graduação em Engenharia Mecatrônica, por lutarem diariamente pelo conhecimento e por um futuro melhor.

Ao professor Dr. José Jean-Paul Zanlucchi de Souza Tavares, pelo incentivo, inspiração, conselhos e por fazer a diferença no quadro docente, com sua visão de mercado e pensamento fora da caixa.

"To think is easy, to act is hard. The hardest is to act in accordance with your thinking."

Johann Wolfgang von Goethe

RESUMO

As tecnologias da indústria 4.0 integram dispositivos e dados, trazendo flexibilidade e eficiência, derivada da decentralização das fontes de informação e processamento, que é fundamental para o avanço das aplicações. Esse trabalho busca introduzir sistemas ciber físicos trabalhando com objetos passivos com capacidade de afetar decisões, um sistema com informação distribuida, e planejadores automáticos, para alcançar um manipulador de processos autossuficiente que não requer inserção externa de objetivos e que pode auto-ajustar-se de acordo com exceções, em tempo de execução. Essa é uma solução mais flexível e autônoma que o uso de máquinas de estado. Aplicando as tecnologias introduzidas na Indústria 4.0 e métodos que eram tratados em separado, como inteligência artificial simbólica e cinemática de robôs, o sistema pode realizar os processos de percepção, planejamento e atuação. Esse sistema é capaz de extrair informação de entidades passivas no domínio físico utilizando Identificação por Rádio Frequência (RFID) para adquirir predicados, dados, sobre os estados corrente e objetivo de cada objeto através da ferramenta PRD (Predicados dentro de base de Dados RFID). Esses dados são tratados para produzir um retrato do domínio, através da união da informação distribuída e produção de uma definição de problema usando a metodologia GISP (Predicados de Estado Individual Agrupados). Essa definição de problema pode ser alimentada no módulo de planejamento, implementado num servidor local ou na nuvem, onde planemento de ações discretas e trajetória são concatenados para retornar referências de controle, usando um planejador simbólico genérico e um gerador numérico de trajetória. Então, o agente ativo pode atuar, verificar exceções e atualizar a informação nos objetos passivos caso o estado obtido seja percebido livre de exceções, senão deve reiterar até que se satisfaça o objetivo global. Esse trabalho estrutura a arquitetura de controle adaptativa a eventos discretos com base de dados RFID contendo partes de lógica de predicados.

Palavras-chave: Sistemas Ciber Físicos, Planejamento Automático, Aplicação de Tempo Real, Cloud Computing, Controle a Eventos Discretos Adaptativo, RFID, PRD, GISP.

ABSTRACT

Industry 4.0 technologies integrate devices and data, bringing flexibility and efficiency, derived from decentralization of information sources and processing, which is fundamental to further advance applications. This work aims to introduce Cyber Physical Systems (CPS) working with decision affecting passive objects, a system with distributed data, and automatic planners, to achieve a self-sufficient process manipulator that does not require external goal insertion and can self-adjust given an exception, in real-time. This solution is more flexible and autonomous than state machines. Applying the technologies introduced in Industry 4.0 and methods that were previously treated separately, such as symbolic artificial intelligence and robot kinematics, the system can perform perception, planning and actuation processes. This system is capable of extracting information inside passive passive entities in the physical domain by using Radio Frequency IDentification (RFID) to acquire predicates, data, about each object current and objective states using the Predicate inside RFID Database (PRD) tool. This data is treated to produce a domain snapshot, by joining distributed information and generating a problem definition, through the Grouped Individual State Predicates (GISP) methodology. This problem definition may then be fed into a planning module, implemented on an Edge or Cloud server, where discrete-action and trajectory planning are concatenated to output control references, using a generic symbolic planner and a numeric trajectory generator. Then, the active agent may actuate, verify for exceptions and update the passive objects information if the obtained state is perceived with no exceptions, else it must reiterate to satisfy the global goal. This work structures the adaptive discrete event control architecture with a RFID database containing parts of predicate logic.

Key-words: Cyber Physical Systems, Automatic Planning, Real-Time Application, Cloud-Computing, Adaptive Discrete Event Control, RFID, PRD, GISP.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

API	Interface de Programação de Aplicações
AWS	Amazon Web Services
CPS	Sistema Ciber Físico
DGM	Modelo Geométrico Direto
EC2	Computação Elástica em Nuvem
FIFO	Primeiro a Entrar Primeiro a Sair
GISP	Predicados de Estado Individuais Agrupados
GNU	GNU Não é UNIX
GPL	Licença Pública GNU
HTTP	Protocolo de Transferência de Hipertexto
IBM	International Business Machines
IGM	Modelo Geométrico Inverso
IP	Protocolo de Internet
IPC	Competição de Planejamento Internacional
iPNRD	PNRD invertida
IIoT	Internet das Coisas Industrial
IoT	Internet das Coisas
JSON	Notação de Objeto JavaScript
LIFO	Primeiro a Entrar Último a Sair
MIMO	Múltiplas Entradas Múltiplas Saídas
NIST	Instituto Nacional de Padrões e Tecnologia
NP	tempo Polinomial Não determinístico
PDB	Bases de Dados de Padrões
PDDL	Linguagem de Definição de Domínios de Planejamento

PHP	Páginas de Hipertexto Pré-processadas
PID	Proporcional Integral Derivativo
PN	Rede de Petri
PNRD	Rede de Petri inserida em base de Dados RFID
POSIX	Interface Portável de Sistema Operacional
PRD	Predicados inseridos em base de Dados RFID
PSS	Espaço de Estados Físico
QR	Resposta Rápida
RFID	Identificação por Rádio Frequência
ROS	Sistema Operativo Robótico
TCP	Protocolo de Controle de Transmissão
UML	Linguagem de Modelagem Universal
USB	Porta Serial Universal
XML	Linguagem de Marcação eXtensível

SUMÁRIO

1 INTRODUÇÃO

Sistemas Ciber-Físicos (CPS) são um dos pilares da Indústria 4.0, no sentido de que o manejo da informação, tomada de decisão e sistemas distribuídos se tornaram profundamente embutidos nos sistemas automatizados (Jazdi, 2014). Uma mudança de paradigma vem acontecendo, onde conectividade, largura de banda, poder computacional e informação cresceram significativamente nos anos recentes (VUKSANOVIC; UGARAK; KORCOK, 2016). Com maior poder de processamento e informação, sistemas mais flexíveis são factíveis, buscando produção personalizada em massa (ROJKO, 2017), com suporte da Internet das Coisas (IoT) e Computação em Nuvem, muitas indústrias e áreas técnicas estão se adaptando na direção de tomada de decisão informada e inteligente. As melhorias recentes estão para ser mais desenvolvidas para aplicação e buscar o aumento de autonomia de forma que haja uma redução da presença humana na produção, reduzindo problemas da saúde na sociedade pós pandemia.

Os campos de robótica e planejamento automático deverão passar por adaptações para estarem aptos à sociedade pós-pandêmica, explorando o crescimento de poder nos CPS, especialmente robôs produtivos. Soluções como máquinas de estado não são suficientes, já que a flexibilidade e alto nível de autonomia agora são requisitos básicos. Suportadas por poder de cálculo e bancos de dados, os CPS devem se tornar mais autônomos mesmo sob problemas de rede.

Diferentes técnicas para tomada de decisão, baseadas em aprendizado por reforço, em planejamento automático clássico, em métodos consumidores de dados, e outros estão sendo pesquisados e testados. Hofer (2017) usa o processo de decisão Markoviano para problemas complexos relacionados com futebol de robôs, e Guerin et al. (2018) usa redes neurais profundas para processar imagens e agrupá-las para ordenamento. Ambas técnicas requerem muito poder computacional que não estava disponível até recentemente.

Em Michniewicz e Reinhart (2016) células robotizadas são virtualizadas e otimizadas de acordo com o fluxo produtivo e a cinemática, usando uma metodologia fora do tempo de execução. Em Xue et al. (2009) é modelada cinemática, dinâmica e propriedades de contato para robôs pegadores para melhoria da pegada de objetos diversos, em tempo de design.

Tavares e Souza (2019) introduzem a integração entre dados PNRD/iPNRD (Redes de Petri Dentro de Base de Dados RFID e PNRD inversa) para resolver o Mundo de Blocos com três blocos numa arquitetura de controle a eventos discretos adaptativa. Os dados PNRD são mantidos em cada entidade passiva por cartões RFID e acessados via leitor RFID no agente ativo. Quando esses componentes são combinados, os estados

iPNRD atuais e objetivo podem ser determinados, assim como o posicionamento físico no Espaço de Estados Físico (PSS, do inglês *Physical State Space*) representado por um espaço de Petri. Uma busca em largura (RUSSELL; NORVIG, 2010) foi implementado para encontrar as transições iPNRD necessárias para alcançar o estado final, gerando controle. Essa abordagem não lida com domínios nos quais a geração completa do espaço de estados é proibitiva. Nesse caso se faz necessário o uso de ferramentas mais poderosas, como planejadores automáticos. Apesar da abordagem PNRD/iPNRD ser útil na redução do envolvimento humano, ela ainda é baseada num espaço de estados fixo, gerado offline.

IoT e Cloud Computing fizeram com que atuação usando planejamento contínuo e online fosse possível para CPS, conforme demonstrado por Fonseca et al. (2016), onde planejamento automático integrado com controladores lógico programáveis é discutido. Ainda assim, um software intermediário para diferentes funções foi necessário, para conectar planejamento e ação num nível de abstração compreensível para um supervisor técnico.

Um problema aberto em planejamento automático é que diversos projetos subestimam a importância e dificuldade de ação deliberativa (NAU et al., 2015) em aplicações online reais, onde agir sob planos abstratos e derivar ações refinadas concretas é computacionalmente demandante. O sistema de planejamento automático clássico é descrito na Figura 1. Este recebe informação externa sobre os estados atual e objetivo do domínio, de forma que o planejador principal deve receber tais dados como entrada de um processo prévio. O planejador alimenta um comando para o controlador atuar sob o sistema. O controlador é retroalimentado já que o resultado do plano pode ser perturbado por eventos externos, para que o plano possa ser refeito para lidar com alterações. Essa arquitetura conceitual é adaptativa por natureza. Esse diagrama apresenta claramente, que, o foco atual da pesquisa em planejamento automático é no planejamento em si (GHALLAB; NAU; TRAVERSO, 2004), já que a percepção e deliberação não é explorada profundamente pela academia.

Esse trabalho mostra como planejamento pode ser modelado, adaptado e ofertado como um serviço, usando uma abstração de nível compreensiva para resolver um problema que se aproxima em complexidade de casos reais, combinado com dispositivos RFID. A ideia de identificar o estado desejado de cada objeto é baseada em embarcar predicados em cartões RFID. Esses predicados podem ser individualmente acessados e podem ser integrados no Agrupamento de Predicados de Estado Individual (GISP do Ingês *Grouped Individual State Predicates*), que é usada para criar o snapshot (cenário do instante) de problemas. Essa técnica que integra GISP e tecnologia RFID é chamada de PRD (Predicados inseridos em base de Dados RFID). PDDL associada com PRD gera um conjunto de ferramentas distribuídas, que podem ser usadas para resolver domínios de sistemas multiagente complexos que não são atendidos pela abordagem PNRD/iPNRD.

Esse trabalho demonstra como a PRD integrada com planejamento automático,

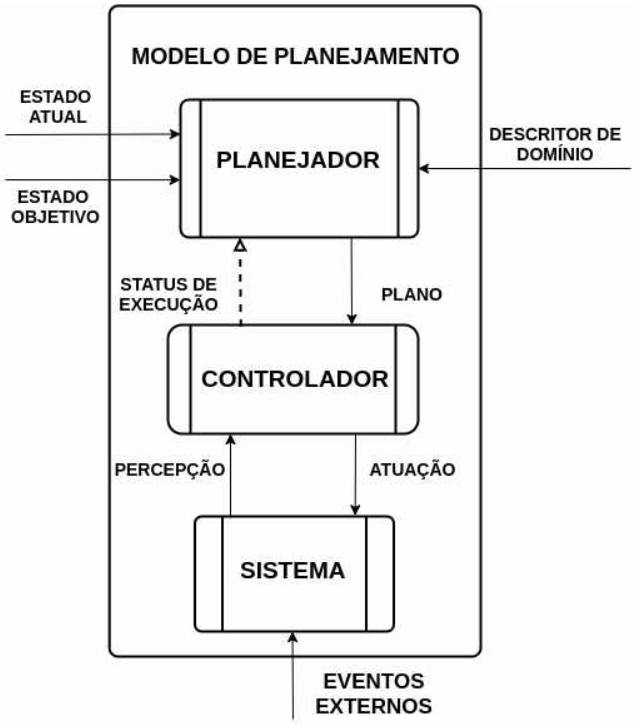

Figura 1 – Modelo Clássico de Planejamento Automático (NAU et al., 2015)

ofertado como serviço, pode assistir no controle adaptativo sob um PSS. Uma variante do mundo de blocos é apresentada como exemplo de aplicação. As próximas secções irão apresentar a base teórica, seguida das ferramentas e arquitetura. Depois são tratadas as implementações e exemplos com resultados. Por fim, são apresentadas conclusões e discussões acerca de trabalhos futuros.

1.1 OBJETIVOS

O objetivo principal é construir ferramentas para a arquitetura de controle a eventos discretos adaptativa proposta nesse trabalho, usando lógica distribuída, planejamento automático de ação e movimento, e conceitos da Indústria 4.0, aplicados num exemplo no Mundo de Blocos. Os objetivos específicos são:

- Desenvolver conhecimento sobre conceitos e tecnologias da Indústria 4.0, como tecnologias de auto-ID, sistemas ciber físicos, computação em nuvem e local e sistemas distribuidos;
- Desenvolver conhecimento sobre ferramentas de modelagem de sistemas a eventos discretos e domínios, modelagem de software, planejamento automático de ação e robótica;
- Projetar uma arquitetura genérica para controle a eventos discretos adaptativo com parte da lógica de predicados descentralizada em base de dados RFID;

- Desenhar as principais ideias e ferramentas a serem utilizadas na arquitetura, a PRD, PSS, Planejamento Automático de ação e movimento e serviço de planejamento remoto;
- Modelar um problema adequado para planejamento e atuação física, o domínio de Mundo de Blocos;
- Adequar a arquitetura e desenvolver software de percepção e planejamento para o domínio especificado;
- Discutir as implementações exploradas;
- Relatar os pontos para continuação e melhoria em trabalhos futuros.

1.2 JUSTIFICATIVAS

Primeiramente é importante citar a atual condição de vulnerabilidade dos sistemas político, econômico e de saúde da sociedade contemporânea. A crise global devido ao vírus COVID-19, que ocorreu durante o desenvolvimento desse trabalho, tem impactos profundos na oferta de empregos, saúde pública e modo de vida das pessoas. A prospecção de novas tecnologias para a automação aplicada nos meios produtivos, logística e pervasiva no cotidiano tem potencial para acrescentar robustez a esses sistemas, pois reduz a interação humana nos produtos manufaturados. Um efeito esperado da automação generalizada é a mitigação dos efeitos sociais causados por crises políticas e de saúde.

Os sistemas produtivos modernos têm sua operação principalmente baseada em máquinas de estado, e em alguns casos de dispositivos com sensores ricos e dinâmica complexa baseada em técnicas de aprendizado por reforço. O sistema proposto nesse trabalho não requer dados de treinamento, diferindo de abordagem baseada em aprendizagem, e é capaz de lidar com várias situações iniciais, sendo limitado, entre outros, pelos modelos e poder computacional.

Nesse sentido, esse trabalho busca desenvolver uma arquitetura com ganhos de flexibilidade e independência, aplicando ferramentas baseadas em tendências industriais e acadêmicas das novas tecnologias, efetivamente reduzindo os casos de retrabalho e exceção que necessitam do envolvimento de pessoas. Esses ganhos também têm valor de mercado, pois tornam a produção mais competitiva ao reduzir custos com mão de obra.

Outra ideia disruptiva desenvolvida nesse trabalho é a distribuição de lógica pelos objetos do domínio, dotando entidades passivas de influência sobre procedimentos ativos, através do embarque de dados em identificadores eletrônicos usando a ferramenta PRD. A abordagem explorada ainda não é aplicada na indústria em grande escala, devido à complexidade e à ausência de um modelo e implementação robusta para a aplicação e

gerência de tais sistemas. Essa complexidade advém da informação distribuida e do grande volume e diversidade de dados, que devem ser processados em tempo de execução pelos atores.

A justificativa educacional, pelo ponto de vista formativo enquanto engenheiro mecatrônico, é de que o desenvolvimento desse trabalho propiciou o ganho de conhecimento teórico e prático do autor em disciplinas de computação e dinâmica, além de competências de arquiteto de sistemas e pesquisador. Lidar com projetos *open-source* e muitas situações, linguagens e ambientes de trabalho possui um valor formativo bastante significativo.

O projeto está aberto para contribuições e abre espaço para novas pesquisas e implementações para o desenvolvimento de uma arquitetura e ferramentas maduras. Esse desenvolvimento fica a cargo do autor e futuros pesquisadores envolvidos.

2 FUNDAMENTAÇÃO TEÓRICA

Devido a multidisciplinaridade e gama diversa de ferramentas usadas nesse projeto é importante uma fundamentação das principais teorias, tecnologias e ferramentas matemáticas usadas na construção dos modelos e implementação dos módulos arquiteturais.

A definição de agentes passivos e ativos é fundamental para o desenvolvimento das ferramentas distribuidas desse trabalho, estando fortemente conexas ao uso de ferramentas de comunicação entre entidades. Essas definições são listadas abaixo.

- **Agente Passivo:** É um agente que não possui mecanismos para alterar o estado do sistema, sendo fisicamente através de atuadores ou computacionalmente através de métodos. Tem seu estado alterado por meio de um agente ativo.
- **Agente Ativo:** É um agente que possui um ou múltiplos mecanismos para interagir ativamente com o sistema, sendo fisicamente através de atuadores e computacionalmente através de métodos. Pode alterar o estado de outras entidades no domínio, incluindo agentes passivos e outros agentes ativos. É também referenciado como ator.

2.1 AUTO-ID

Procedimentos de identificação e medição automática são de uso comum na indústria, para compra, logística, manufatura, inventário e fluxo, entre outras aplicações. Esses sistemas de identificação automática estão associados a tecnologias que acessam informação provida pela entidade amostrada, na forma de dados etiquetados, ou sistemas de percepção que inferem dados com base em características sensoriadas, como os sistemas de visão.

O código de barras é extremamente popular e se mantém como um dos sistemas mais populares de identificação por décadas. Devido ao seu baixo custo, fácil implantação e uso cotidiano esteve incluso em diversas etapas da cadeia de suprimentos. Porém é problemático devido à baixa capacidade de armazenamento e por não ser reprogramável a nível de objeto (FINKENZELLER, 2010). Outra tecnologia popular na captura de informação mas que sofre dos mesmos problemas é o código QR (do inglês *Quick Response*, Resposta Rápida), mais popular em aplicações Web pois é passível de regeração, utilizável em sistemas de pagamento, referenciamento, como em Dodson et al. (2012).

Sistemas de visão são usados na indústria para percepção de características físicas de objetos e entidades, não para coleta de dados atrelada a objetos com identidade única ou associada a classe, como é o caso das tecnologias códigos de barra e QR, e RFID (Identificação por Rádio Frequência). Podem ser usados de forma automática na detecção

de falhas de fabricação e direcionamento de fluxo, como explorado em Schluter et al. (2018).

Os sistemas RFID usam ondas de rádio-frequência para transferência de dados entre etiquetas, armazém de dados, e leitores, dispositivos de resgate de dados, seguindo princípios de modulação e demodulação de informação digital. Podem operar em baixas distâncias, usando baixa energia e etiquetas passivas (sem armazenamento/uso ativo de energia), ou em médias distâncias usando leitores de alta energia e etiquetas ativas, no estado da arte atual (MUROFUSHI, 2016). São altamente empregados nas indústrias de logística, pecuária, em aplicações de automação de contagem, acesso e fluxo. Possuem alto potencial na manufatura (FINKENZELLER, 2010).

Lotlikar et al. (2013) traz uma análise e comparativo entre as tecnologias de coleta de dados embarcados de código de barras, códigos QR e RFID. As vantagens da tecnologia RFID no ponto de vista de robustez, adaptabilidade, manejo e capacidade de dados fica clara, pois essa não requer linha de visão, possuí distância de leitura adaptável, atribui identificação única aos objetos, é mais fácil de se automatizar que sistemas ópticos devido a robustez de orientação e posição de leitura, tem elevada confiabilidade de captura de dados, e operações de (re)escrita podem ser executadas pelo leitor RFID. Comparado com sistemas de visão, a captura de dados não é baseada em inferência, mas na obtenção direta da informação, o que aumenta a confiabilidade e reduz o custo computacional de processamento.

2.1.1 Radio Frequency Identification (RFID)

Um sistema RFID é composto de dois elementos, etiquetas e antenas, o primeiro associado a objetos passivos e o segundo associado a sistemas de processamento de informação e agentes ativos. A troca de informação ocorre em pares, conforme a Figura 2, enfatizando a troca de dados sem contato.

- **Etiqueta:** Se localiza embarcada/colada no objeto de interesse. É a parte que armazena dados do sistema, contém memória persistente e um transponder. Quando possui bateria, pode-se dizer que é uma etiqueta ativa, e quando não possui, é passiva. Trabalha de forma reativa.
- **Leitor:** Dispositivo ativo que pode ser do tipo leitor ou escritor/leitor, dependendo do modelo. Contém um módulo de *encode/decode* e uma antena, a qual permite comunicação *duplex* com o transponder na etiqueta.

O RFID, somado às tecnologias complementares de sensoriamento e percepção, tem potencial na computação pervasiva, permitindo que sistemas de computação possam observar, identificar e entender o mundo sem supervisão constante ou alimentação de

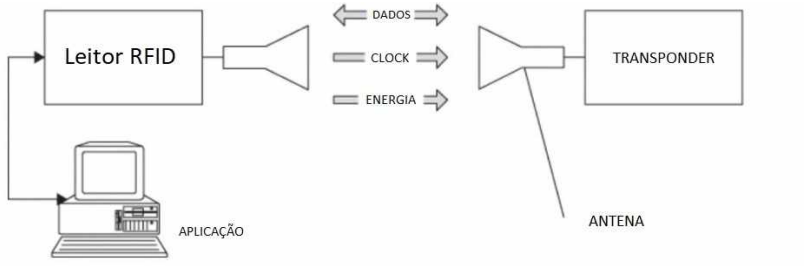

Figura 2 – Comunicação em par RFID

dados por seres humanos (ASHTON et al., 2009). Essa capacidade de propiciar alta adaptabilidade a sistemas distribuídos de objetos no domínio material torna a tecnologia RFID comum no meio IoT (Internet das Coisas) e a escolha natural para esse trabalho.

2.2 REDES DE DISPOSITIVOS

Uma tendência moderna é a conectividade e cooperação entre dispositivos por redes locais e pela Internet. As tecnologias de rede são pervasivas e muitos dos serviços consumidos cotidianamente são associados a redes de sensores e/ou a nuvem, como sistemas de segurança em bancos e o acesso a redes sociais em dispositivos móveis.

2.2.1 Internet das Coisas e de Tudo

A Internet das Coisas, do inglês *Internet of Things* (IoT), é uma tendência da automatização do envio de dados para a internet por uma rede de dispositivos diretamente conectados, sem que haja ação ou supervisão humana direta no envio ou tratamento dos dados. O termo foi cunhado por Kevin Ashton em 1999, numa apresentação na Procter & Gamble.

Desse ideal de conectividade direta surge a ideia de que a comunicação com as "coisas" traz mais informação aos computadores, tornando as soluções mais precisas e reativas (ASHTON et al., 2009). Atualmente diversas aplicações de IoT estão se difundindo, devido aos avanços tecnológicos e redução de custos. Aplicações como casas conectadas, agronegócio avançado e indústrias inteligentes, se tornam mais comuns (SILVA et al., 2016).

A Internet de Tudo, assim denominada pela Cisco, é também referenciada como *Industrial IoT* (IIoT). A IIoT embasa decisões no chão de fábrica com base nos dados dos equipamentos, sensores, produtos e informações externas, como dados gerenciais (GILCHRIST, 2016). Nessa abordagem as tecnologias de processamento de dados e tratamento de objetos distribuídos se tornam pervasivas no meio industrial, fazendo com que conceitos como *Middlewares, Big Data* e sistemas remotos (COULORIS; DOLLIMORE;

KINDBERG, 2007) baseados em dados adentrem na indústria tradicional, de forma que as responsabilidades de processamento de dados possam ser delegadas a máquinas remotas.

Com o aumento da base de dados e da tomada informada de decisão há potencial para revolucionar a forma com que engenheiros, analistas e gestores trabalham, fazendo com que a retroalimentação entre os setores produtivos e corporativo das empresas seja mais ágil e eficaz. Esse conceito pode causar a instauração de um novo paradigma de trabalho e social (LANGLEY et al., 2020), fortemente estruturado sobre as infraestruturas em nuvem.

2.2.2 Computação em Nuvem e Indústria

A computação em nuvem oferece tipos de serviços que estão ganhando força na IoT e IIoT (Pan; McElhannon, 2018). De acordo com o NIST, o Instituto Nacional de Padrões e Tecnologia do Departamento de Comércio dos Estados Unidos, em Mell, Grance et al. (2011), a computação em nuvem é um paradigma em evolução contínua caracterizado por habilitar a computação ubíqua, conveniente e sob demanda de recursos e serviços computacionais acessados pela rede. Tem as características de provisão rápida, elasticidade e esforço administrativo mínimo. Suas características essenciais podem ser listadas:

- **Autosserviço sob Demanda:** Os clientes têm capacidades unilaterais de selecionar recursos de processamento, tempo de serviço e armazenamento de forma automática, sem necessidade de interagir com pessoas do provedor do serviço;
- **Acesso em Rede Generalizado:** Qualquer tipo de cliente pode acessar os recursos por mecanismos de rede e conexão padronizados;
- **Gestão Dinâmica de Recursos:** O provedor de serviços pode realizar alocação de recursos de forma dinâmica, de acordo com a necessidade dos clientes. O ponto físico de alocação dos recursos tem pouca importância para o cliente e é tratado de forma abstrata ou transparente;
- **Elasticidade:** O provedor de serviços tem a capacidade de prover os recursos adequados para demandas crescentes, até de forma automática, criando a ilusão de recursos ilimitados para o cliente;
- **Instrumentação de Serviços:** O custo computacional dos serviços é medido, sendo reportado, otimizado e taxado de forma apropriada.

Os principais provedores de serviço em nuvem, em 2020, são corporações como a Amazon, Microsoft, Google e IBM. As formas de oferta são três: infraestrutura, plataforma e aplicação (BHARDWAJ et al., 2010). Na primeira os recursos de hardware são diretamente

oferecidos ou é dado acesso a uma máquina virtual. Na segunda é oferecido um *container*, com ferramentas, serviços-escravos e recursos para desenvolvimento de aplicações. Na terceira é oferecido diretamente um software ou microsserviço. As formas de acesso podem ser por meio de uma nuvem pública, privada ou de acesso híbrido, dependendo da aplicação do contratante. Os consumidores desses serviços são pessoas, e organizações de todos os tamanhos que as utilizam para serviços como hospedagem de repositórios, bancos de dados e serviços de software.

As principais formas para acesso remoto e acesso local do serviço usadas na indústria são através de tecnologias de comunicação Web, em nuvem e tecnologias de comunicação de proximidade, em especial cabeada (Aijaz; Sooriyabandara, 2019). Cada arquitetura, protocolo e meio de transmissão tem características específicas de banda, determinismo, latência, segurança, garantia de entrega, imunidade a ruído e custo de implantação. Galloway e Hancke (2013) trazem uma discussão rica sobre o ecossistema de redes industriais. Com relação a custo e velocidade, qualquer pode ser mais eficiente entre *Cloud* (nuvem) e *Edge* (borda), dependendo do custo computacional do problema, condições da rede e poder de processamento do servidor. Ai, Peng e Zhang (2018) apontam que *Cloud Computing* foi consolidada como altamente eficiente e flexível, numa arquitetura centralizada na nuvem. Contudo, devido ao aumento na presença de aplicações IoT, suas restrições e problemas derivados causam a ineficiência do paradigma de servidor central. Portanto soluções Edge e híbridas são investigadas pela academia. Uma abordagem seria hospedar o planejador na nuvem, e ter uma máquina local na borda como redundância ou reserva, no caso de latência elevada ou indisponibilidade. Isso produz um mecanismo protetivo, para garantir operação mesmo sob falhas na rede.

Nesse trabalho, para a construção do processo foi usado o serviço de computação elástica em nuvem AWS EC2©, do inglês *Elastic Compute Cloud*, um serviço de infraestrutura adaptável (KULKARNI et al., 2011) no modo de acesso privado. Nesse serviço é possível escolher a imagem do sistema operacional, o hardware, número de núcleos, política de divisão/exclusividade de recursos, memória primária e secundária nas versões pagas. Na versão experimental, a utilizada, as especificações são como descritas na segunda implementação na Secção 4.2.6.

2.3 LINGUAGENS

Linguagens de representação são fundamentais para modelagem de sistemas e auxiliar na implementação de soluções. Nessa secção são tratadas linguagens gráficas, de matemática discreta e de representação lógica de informação.

2.3.1 Máquinas de Estado Finito

Máquinas de Estado Finito são ferramentas de modelagem matemática de sistemas a eventos discretos. São populares em disciplinas de ciência da computação e engenharia elétrica, por serem capazes de modelar estados e fluxo segundo uma notação simples, algébrica ou gráfica. São usadas como referência na modelagem de sistemas com espaço de estados discreto, na implementação de software/firmware e da caracterização de sistemas de comportamento variantes no tempo ou híbridos, como no trabalho de Yi-Liang Chen e Feng Lin (2000), onde uma metodologia de modelagem e um exemplo em logística são abordados.

As Máquinas de Estado Finito são, nesse trabalho, a abordagem mais simples em termos de modelagem e implementação para tratar de processos como identificação do espaço de estados, e, em alguns casos, as representações em modelos mais complexos como redes de Petri podem ser simplificadas. O resultado dos processos de planejamento é interpretado como uma máquina de estado finito, com processamento sequencial pelo módulo executor.

2.3.2 Redes de Petri

De acordo com (Murata, 1989), Redes de Petri (RP) consistem de representação gráfica e modelo matemático. RP providenciam primitivas básicas para modelagem de concorrência, comunicação e sincronização. Formalmente uma RP elementar é definida como a quádrupla $RP = (P, T, F, M_0)$, em que P é um conjunto finito de posições, T um conjunto finito de transições, onde $P \cap T = \emptyset$, $F \subseteq (TxP) \cup (PxT)$ é a relação de fluxo, o conjunto de arcos, e M_0 é a marca inicial. RP podem ser representadas matematicamente como uma Equação Matricial (Equação 2.1), onde M_{k+1} é a próxima marcação, M_k é a marcação corrente, A^t é a matriz de adjacência e u_k é o vetor de disparo.

$$M_{k+1} = M_k + A^t \cdot u_k \tag{2.1}$$

As RP simples são úteis na modelagem de sistemas dinâmicos determinísticos a eventos discretos, podendo representar espaços de estado e transições abstratas e serem usadas como estrutura de busca (SOUZA; BRITO; TAVARES, 2019), em associação a outros formalismos e tecnologias (BARRETO, 2019).

2.3.3 Diagramas UML

A Linguagem de Modelagem Universal, do inglês *Universal Modeling Language* (UML) é uma linguagem gráfica diagramática para especificação de comportamento e estrutura de módulos de sistemas que se baseia principalmente nos conceitos de orientação

a objetos (FOWLER, 2003). São ferramentas comuns nas disciplinas de engenharia de software e engenharia de conhecimento, para auxiliar na modelagem de domínios e sistemas (VAQUERO, 2007).

A estrutura funcional dos principais diagramas UML pode ser vista na Figura 3. Os tipos de diagramas separam-se basicamente em diagramas estruturais e comportamentais, cada um com regras específicas de notação acordadas pela literatura. Cada tipo é útil numa etapa distinta de modelagem, implementação e documentação de projetos.

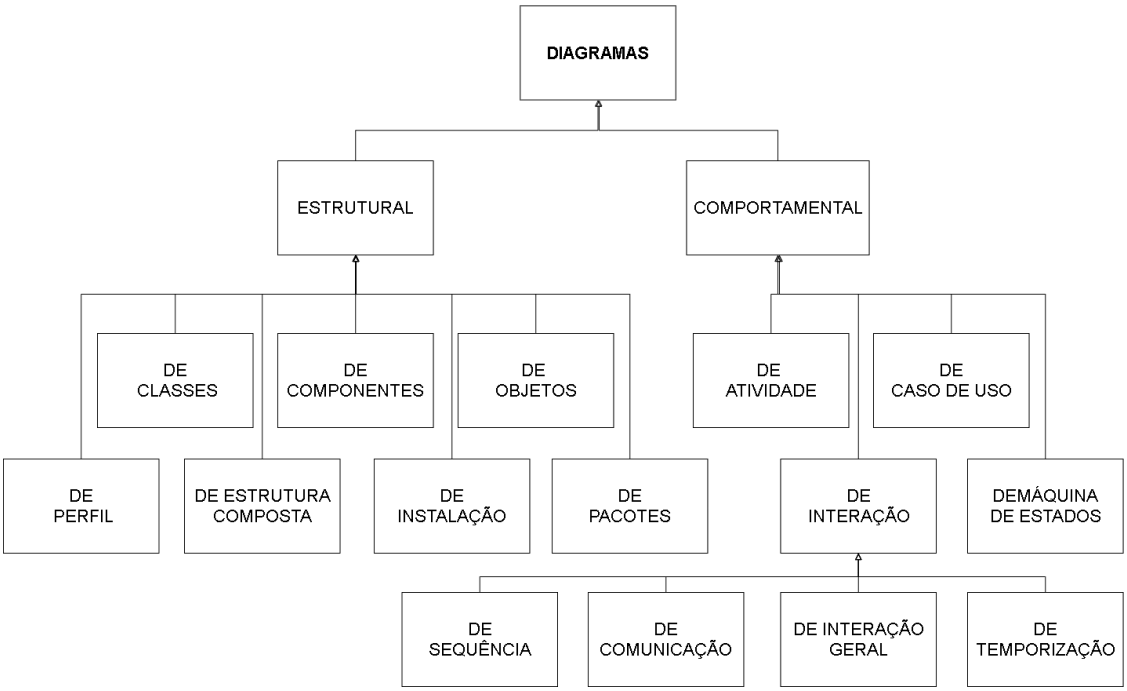

Figura 3 – Árvore de Diagramas UML

UML é usado nesse trabalho para construção do modelo de domínio do estudo de caso abordado. A construção de um bom modelo é vital para a boa adaptação da arquitetura de controle, performance da implementação, e comportamento adequado/suficientemente preciso, tendo em vista que a implementação dos planos e processos são acoplados ao modelo de domínio.

2.3.4 Predicados

Predicados são elementos que tratam do estado/característica de um objeto/classe ou relacionamento entre objetos/classes segundo a lógica de predicados (NUNES, 2020), onde é possível inferir informação ou avaliar se um predicado é verdadeiro ou falso dado contexto suficiente, na forma de um banco de predicados.

São base para linguagens lógicas (EMDEN; KOWALSKI, 1976), como o PROLOG e dos modelos representativos para planejamento automático, que se fundamentam no paradigma lógico de programação. O paradigma lógico tem escopo e aplicação diferente

dos outros paradigmas de computação (BARANAUSKAS, 1993), sendo caracterizado pela existência de uma base de conhecimento e pela declaração de "dúvidas", onde o processo de inferência (procedural) é abstraído da declaração do programa. O paradigma lógico é um subparadigma declarativo, onde o programador não se preocupa diretamente em explicitar a execução dos procedimentos, mas em descrever o domínio e problema.

As principais aplicações da programação lógica na indústria são para resolução de problemas fora de tempo de execução, no planejamento de tarefas e alocação de recurso em tempo offline e de design. Em Meneghetti (2009) programação lógica é usada para otimização da alocação de recursos num armazém. Na academia a programação lógica é usada, por exemplo, para a prova de teoremas matemáticos (FELTY; MILLER, 1988). Nesse trabalho os predicados são base construtiva para a especificação da PRD e são elemento constitutivo da PDDL.

2.3.5 PDDL

PDDL é uma linguagem textual formal usada para descrever domínios (MCDERMOTT et al., 1998), e também como padrão de entrada para planejadores, como um descritor de sistemas portável. É usada nesse trabalho para entrar com informação nos serviços de planejamento.

A maioria dos planejadores genéricos modernos são capazes de utilizar a PDDL como representação de entrada do domínio para a geração de um plano de ação já que esta linguagem tornou-se um padrão na área de Planejamento Automático. A representação do modelo do domínio deve ser adequada, suficientemente próxima da realidade, contendo a descrição das ações possíveis no domínio, suas pré e pós-condições, as informações sobre o estado inicial do domínio e o estado objetivo (junção de metas) para que o planejador possa gerar a estrutura de busca e buscar pela solução adequada (VAQUERO, 2007). Algumas das características da PDDL são:

- A PDDL é um modelo direcionado às ações do domínio e suas ações são baseadas em ações do modelo STRIPS (FIKES; NILSSON, 1971) onde as pré-condições e efeitos de uma ação representam a dinâmica da execução dessa no domínio;
- Especificação de ações hierárquicas compostas por sub-ações;
- Definição de restrições;
- Diversos outros recursos associados a versão utilizada.

PDDL é uma linguagem em evolução constante para ganho de expressividade, com atualizações frequentes e algumas derivações para inclusão de recursos específicos. Uma

referência moderna é HASLUM et al. (2019). A versão para referência usada nesse trabalho é a PDDL 3.0.

2.4 PLANEJAMENTO AUTOMÁTICO

Planejamento automático é a geração de planos de ação baseadas na abstração de um modelo de domínio e problema numa estrutura de busca, como um grafo, e da identificação de um plano de transições/estados intermediários que leva o sistema do estado corrente até o conjunto de objetivos.

O planejamento automático é um campo de inteligência artificial fortemente teórico (GHALLAB; NAU; TRAVERSO, 2014), mas com potencial de aplicação genérica em problemas do mundo real. Esses planejadores são capazer de encontrar soluções adequadas para problemas complexos, desde que os modelos sejam adequados e haja poder computacional suficiente (MARTINEZ; SILVA, 2019) para caracterizar boas soluções em tempo hábil. Em aplicações de robótica o uso de planejadores genéricos vem ganhando espaço, como da inclusão de um pacote de planejamento de ação baseado em PDDL no ROS, o Sistema Operativo Robótico (Silva Miranda; de Souza; Sousa Bastos, 2018).

O uso de algoritmos de busca informada e heurísticas eficientes pode reduzir o custo computacional da busca e melhorar o plano resultante, com garantias de planejamento ótimo, caso seja essencial.

2.4.1 Algoritmos de Busca em Grafo

Algoritmos de busca são técnicas para expansão do grafo de estados explorados, sendo determinante forte na geração de um plano de ação. Em sistemas de planejamento as técnicas usuais são determinísticas e completamente baseadas em modelo. A busca pode ser feito partindo do estado corrente até o estado objetivo, *forward*, ou do estado objetivo até o atual, *backward*.

Russell e Norvig (2010) traz uma descrição das propriedades de algoritmos de busca, onde completude, complexidades de tempo e espaço mínimas, e geração de plano ótimo são desejáveis na maioria dos problemas. Essas propriedades e características são listadas abaixo.

- **Completa:** Um algoritmo de busca é dito completo se dado um tempo infinito de busca ele é capaz de encontrar qualquer solução alcançável.
- **Ótima:** Um algoritmo é dito ótimo se ele sempre encontra primeiro um plano de ação com custo mínimo, isto é, ótimo.

- **Complexidade de tempo e espaço:** Algoritmos de busca acoplados a suas subespecificações e ao problema alvo tem complexidade de tempo e espaço teóricas atreladas. Na maior parte dos problemas de busca, independente da técnica usada, a complexidade é exponencial no pior caso, dada a natureza *NP-hard* da maioria dos problemas (GIMENEZ; JONSSON, 2008).

Existem duas principais famílias de algoritmos de busca para espaços de estados discretos, os algoritmos cegos e informados.

Os algoritmos cegos são buscas que não utilizam informação-guia sobre o domínio, heurísticas, para traçar sua expansão de estados. Na prática exploram o espaço de estados por força bruta, buscando a solução. Existem duas técnicas principais: busca em largura e busca em profundidade.

- **Busca em Largura:** Nessa busca todos os nós em um determinado nível de profundidade da árvore são expandidos, até exaurir as possibilidades para até aquele nível. Esse algoritmo, supondo custo de ação homogêneo, é ótimo e, dado tempo de execução infinito, sempre encontra a solução. É implementado por meio de uma fila FIFO, do inglês *First In First Out*, o que significa que conforme os estados são expandidos os novos estados entram para o fim da fila, de forma que os mais antigos sejam expandidos primeiro.
- **Busca em Profundidade:** Na busca em profundidade a exploração é implementada por meio de uma pilha, ou fila LIFO, do inglês *Last In First Out*, o que significa que os estados expandidos mais recentes entram no topo da fila de expansão. Não há garantias de plano ótimo. Ou mesmo de completude, caso a profundidade seja infinita.

Diferindo de algoritmos cegos, os algoritmos informados usam algum conhecimento para guiar a expansão dos nós. Esse conhecimento é advindo da heurística, que é uma quantificação da expectativa de proximidade da solução. Heurísticas são específicas de problemas ou geradas com base na estrutura genérica de busca. Os algoritmos informados implementam filas de expansão prioritária, usando da política específica do algoritmo e do valor de heurística para calcular o ponto de inserção dos novos nós gerados na fila de expansão. A política mais comum é minimização do custo esperado, sendo o custo calculado por uma equação. Citam-se dois algoritmos informados básicos: a busca "gulosa" e a busca A*.

- **Busca "Gulosa":** A busca "gulosa", ou gananciosa, é um algoritmo com tendências comportamentais de se mover adiante, sem retornar e expandir nós mais rasos. A função custo (Equação 2.2) é dada apenas pela função heurística $h(n)$, onde o custo

$f(n)$ é o custo esperado até a solução. Essa função favorece a expansão de nós nas fronteiras mais profundas, sendo "gulosa" pois tem uma tendência a se mover em profundidade. A busca gulosa é incompleta e não ótima, sendo uma alternativa ágil para determinar planos de ação, mas sem garantia de plano ótimo.

$$f(n) = h(n) \tag{2.2}$$

- **Busca A*:** Na busca A* há a adição de um termo de custo real até a posição atual $g(n)$, conforme a Equação 2.3 o que adiciona um efeito de memória e favorece alguma expansão de nós rasos. Esse algoritmo de busca é ótimo e completo para heurísticas admissíveis. Algoritmos mais eficientes que o A* são, em sua maioria, derivações que se aproveitam de características da família do domínio para gerar otimizações, ou junções com outros algoritmos advindos de uma família de técnicas distinta (KUMAR et al., 2011).

$$f(n) = g(n) + h(n) \tag{2.3}$$

Os algoritmos de busca informada baseados em A*, usando heurísticas admissíveis, sempre encontram a solução ótima e possuem complexidade computacional de tempo e espaço inferiores, quando comparados a métodos não informados. A seleção de um algoritmo adequado é essencial, pois na maioria dos problemas, no pior caso, a complexidade de busca ainda é exponencial. O acoplamento com a heurística adequada também é muito importante para minimizar o número e os custos de expansões.

2.4.2 Heurísticas

Heurísticas são expectativas da proximidade da solução, concebidas através do relaxamento do problema de busca ou da observação de subproblemas. Podem ser desenhadas especificamente para a família do domínio alvo ou auto-geradas pelo planejador dadas as características do problema. São esses os tipos especialista e baseada em subproblemas (experiência).

Heurísticas especialistas são, por exemplo, a heurística de distância euclidiana ou da distância de Manhattan usadas para problemas de planejamento de trajetória por robôs móveis (KUMAR et al., 2011). Das heurísticas auto-geradas podemos citar o trabalho de PRIEDITIS (2008), onde heurísticas são obtidas através da experiência de solução de subproblemas e podem ser combinadas em tempo real, usando uma abordagem de *Multiple Pattern Databases*, onde a heurística adequada ou pré-computação é usada na solução dos subproblemas com determinadas características/padrões. Essa heurística é conhecida na literatura como *PDB*, do inglês *Pattern DataBase*.

Propriedades de heurísticas: (RUSSELL; NORVIG, 2010)

- **Admissível:** uma heurística é admissível se $h(s) \leq h^*(s)$ para todo nó s. O que significa que a estimativa de custo para alcançar o objetivo deve sempre ser uma subestimativa ou ser igual ao valor de custo real $h^*(s)$. Estimativas superiores ao custo real tornam a heurística inadmissível, o que pode causar falhas de convergência do algoritmo de busca.
- **Consistente:** uma heurística é consistente se $h(s) \leq c(s, s') + h(s')$ para todos os estados s conectados a um estado s' por uma ação de custo $c(s, s')$. Isso significa que a estimativa de custo original deve ser sempre inferior ao custo real da transição acrescido da nova estimativa, para que respeite a propriedade de admissibilidade e a desigualdade triangular.
- **Segura:** Uma heurística é segura se $h(s) = \infty$ é verdadeira apenas para $h^*(s) = \infty$. Segurança significa que o custo estimado da heurística é infinito apenas se o estado objetivo for inalcançável ou estiver infinitamente distintante. Heurísticas admissíveis são seguras, mas a recíproca não é necessariamente verdadeira.

Heurísticas podem apresentar relacionamento de dominância. Uma heurística h^1 é dita dominante sobre h^2 se $h^1(s) \geq h^2(s)$ para todo nó s e ambas são admissíveis. Isso significa que a estimativa de custo gerada por h^1 é mais próxima do custo real, melhor que h^2, consequentemente gerando menos estados expandidos e acelerando a convergência da busca (RUSSELL; NORVIG, 2010).

Na prática algumas heurísticas só devem ser aplicadas na presença/ausência de características de modelagem de domínio do sistema, para que mantenham suas propriedades. Algumas implementações de heurísticas não se comportam bem na presença de custos variados de ações, axiomas e efeitos condicionais, entre outras características de modelagem de domínio, conforme relatado na documentação oficial do planejador *Fast-Downward* (DEVELOPERS, 2018), logo deve-se evitar o uso de heurísticas frágeis ou modelar o domínio de forma a minimizar/eliminar a presença de características especiais, que ferem o uso da heurística alvo.

2.5 ROBÔS MANIPULADORES

Para se trabalhar com robôs manipuladores são necessários modelos geométricos e políticas de trajetória. As subsecções nesse capítulo tratam, em ordem, da modelagem geométrica do robô e das políticas de otimização e geração de trajetória robótica adotadas.

2.5.1 Modelo Geométrico Direto

O Modelo Geométrico Direto (DGM) é uma representação das equações cartesianas de um robô manipulador que relaciona a posição e orientação de um ponto na estrutura robótica e suas variáveis articulares (LYNCH; PARK, 2017). Em robótica o principal ponto estrutural para planejamento de ação é o elemento terminal, normalmente um efetuador, como uma garra, bico de solda ou lâmina, que usualmente se encontra no extremo da cadeia articular do robô. Logo o DGM é fundamental no cálculo direto de entradas articulares para alcançar posições no espaço de trabalho.

Um robô com estrutura em árvore é composto de n articulações atuadas. Portanto a conversão de referência pode ser feita como um produto de n matrizes homogêneas de transformação. Essas matrizes podem ser obtidas usando métodos geométricos de modelagem ou outras técnicas (ANGELES, 2002). A expressão matricial que representa a concatenação das transformadas é dada na Equação 2.4.

$$ {}^{0}T_{n[4\cdot4]}(q_m) = {}^{0}T_1 \cdot {}^{1}T_2 \cdots {}^{n-1}T_n \tag{2.4} $$

Onde q_m representa as entradas articulares, com m entradas. A variável $n \in \mathbb{N}^*$, sendo um número arbitrário que representa o número de transformadas usadas, normalmente associado ao número de articulações ou atuadores. $m \in \mathbb{N}^*$ representa o número de atuadores efetivos do sistema. ${}^{0}T_{n[4\cdot4]}$ representa a transformada que converte entradas articulares, seus parâmetros, na matriz de transformação direta, com dimensão $4x4$ para embutir rotação e translação (LYNCH; PARK, 2017).

X_i representa a saída cartesiana com i variáveis. A variável $i \in \mathbb{N}^*$ representa o número de dimensões modeladas, podendo ser até 6 para um ponto estrutural, 3 de posição e 3 de orientação, representadas por $X\ Y\ Z$ e $\psi\ \theta\ \phi$, respectivamente posição e rotação em torno dos eixos ortonormais x, y e z. A obtenção de X_i pode ser feita através de ${}^{0}T_n$ pela Equação 2.5 ou de modelagem direta (ANGELES, 2002). Note que não há uma formulação genérica direta para obtenção dos ângulos de Euler (rotações).

$$ \begin{bmatrix} X \\ Y \\ Z \\ \psi \\ \theta \\ \phi \end{bmatrix} = \begin{bmatrix} {}^{0}T_n(1,4) \\ {}^{0}T_n(2,4) \\ {}^{0}T_n(3,4) \\ f_\psi(\psi,\theta,\phi) \\ f_\theta(\psi,\theta,\phi) \\ f_\phi(\psi,\theta,\phi) \end{bmatrix} \tag{2.5} $$

A obtenção genérica dos ângulos de Euler é dada pela conversão da Matriz de Rotação (Equação 2.6) na Equação 2.7, um vetor, e sua diferenciação conforme Equação 2.8, onde pode ser aplicado um algoritmo de minimização de erro, como o método dos mínimos quadrados não linear (WESSTEIN, 2020). Essa formulação genérica tem custo

computacional relativamente elevado, logo expressões específicas são preferíveis caso o modelo DGM do robô seja conhecido.

$$R_{3\cdot3} = \begin{bmatrix} f_1(\psi,\theta,\phi) & f_2(\psi,\theta,\phi) & f_3(\psi,\theta,\phi) \\ f_4(\psi,\theta,\phi) & f_5(\psi,\theta,\phi) & f_6(\psi,\theta,\phi) \\ f_7(\psi,\theta,\phi) & f_8(\psi,\theta,\phi) & f_9(\psi,\theta,\phi) \end{bmatrix} \tag{2.6}$$

$$\left[f_{1\cdot9}\right] = \begin{bmatrix} f_1(\psi,\theta,\phi) \\ ... \\ f_9(\psi,\theta,\phi) \end{bmatrix} \tag{2.7}$$

$$df = \begin{bmatrix} \frac{f_1(\partial\psi,\theta,\phi)}{\partial\psi} & \frac{f_1(\psi,\theta,\phi)}{\partial\theta} & \frac{f_1(\psi,\theta,\phi)}{\partial\phi} \\ ... & ... & ... \\ \frac{f_9(\partial\psi,\theta,\phi)}{\partial\psi} & \frac{f_9(\psi,\theta,\phi)}{\partial\theta} & \frac{f_9(\psi,\theta,\phi)}{\partial\phi} \end{bmatrix} \cdot \begin{bmatrix} d\psi \\ d\theta \\ d\phi \end{bmatrix} \tag{2.8}$$

A definição de $atan2$ é dada pela Equação 2.9 (ORGANICK, 1966), sendo usada na solução prática dos ângulos de rotação, através da análise do DGM.

$$\operatorname{atan2}(y,x) = \begin{cases} \arctan(yx^{-1}) & \text{if } x > 0, \\ \arctan(yx^{-1}) + \pi & \text{if } x < 0 \text{ and } y \geq 0, \\ \arctan(yx^{-1}) - \pi & \text{if } x < 0 \text{ and } y < 0, \\ +0.5\pi & \text{if } x = 0 \text{ and } y > 0, \\ -0.5\pi & \text{if } x = 0 \text{ and } y < 0, \\ \text{undefined} & \text{if } x = 0 \text{ and } y = 0. \end{cases} \tag{2.9}$$

A função do DGM pode ser visualizada pela Figura 4. Alimentando o modelo com os parâmetros articulares é possível calcular as variáveis cartesianas de saída.

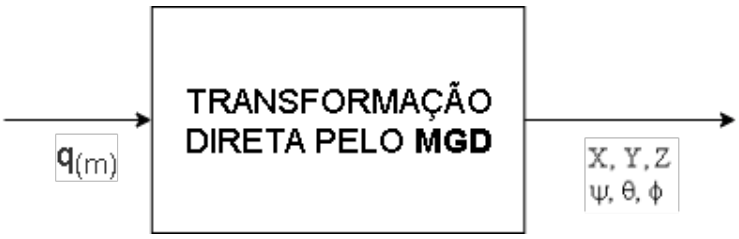

Figura 4 – Visualização da transformação direta no DGM

O GMD é necessário para calcular a saída cartesiana de um sistema, porém é insuficiente para a maior parte das aplicações, que estão interessadas em determinar entradas articulares para alcançar uma saída já determinada. Com o Modelo Geométrico Inverso (IGM) é possível obter soluções para o vetor de entradas (ANGELES, 2002).

2.5.2 Modelo Geométrico Inverso

A abordagem tradicional de cinemática inversa é baseada na inversão analítica do DGM (ANGELES, 2002). Na abordagem inversa, as entradas articulares são determinadas em termos de posição e atitude do elemento terminal (Figura 5), retornando todas as soluções possíveis para um ponto particular no espaço de estados, normalmente feita através de análise geométrica ou inversão manual do DGM.

Figura 5 – Visualização da transformação inversa pelo MGI

A obtenção do MGI é um processo caro do ponto de vista de modelagem e que resulta em equações extensas e com vários termos, sendo necessário refazê-la para cada modelo estrutural de robô, além de sofrer de problemas de singularidade (SANDLER, 1999) e necessidade de política para adoção de uma solução, no caso de múltiplas soluções, o que acontece frequentemente, especialmente em robôs redundantes. Além disso a complexidade escala drasticamente com o número de articulações, comparado ao DGM.

Na vizinhança de pontos de singularidade ocorrem problemas no cálculo das entradas de controle. A interpretação matemática é que a matriz Jacobiana (Secção 2.5.3) perde *rank* e fisicamente que o robô perde um grau de liberdade nesses pontos. Isso explica a necessidade de uso de métodos numéricos, já que o comportamento do robô muda "violentamente" nas proximidades dos pontos de singularidade.

2.5.3 Jacobiano e Inversa de Moore-Penrose

Em robótica o Jacobiano relaciona diretamente velocidades entre o elemento terminal e velocidades articulares (LYNCH; PARK, 2017). A Equação 2.10 demonstra o relacionamento, onde $\dot{X}$ e $\dot{q}$ são respectivamente os vetores de velocidades cartesianas do elemento terminal e articulares, e J é o Jacobiano Geométrico, com n dimensões no elemento terminal e m entradas articulares. Deve ser observado que a matriz do Jacobiano não é necessariamente quadrada, logo pode não possuir inversão direta.

$$\dot{X}_{[n\cdot 1]} = J_{[n\cdot m]}\dot{q}_{[m\cdot 1]} \tag{2.10}$$

Em matemática e otimização (PATRIKSSON et al., 2016), o Jacobiano pode ser escrito como uma matriz de derivadas parciais com relação às variáveis de entrada. A

Equação 2.11 mostra o relacionamento entre variáveis de saída e entrada V_o e V_i no Jacobiano, considerando n saídas e m entradas.

$$J = \begin{bmatrix} \frac{\partial V_{o1}}{\partial V_{i1}} & \frac{\partial V_{o1}}{\partial V_{i2}} & \cdots & \frac{\partial V_{o1}}{\partial V_{im}} \\ \frac{\partial V_{o2}}{\partial V_{i1}} & \frac{\partial V_{o2}}{\partial V_{i2}} & \cdots & \frac{\partial V_{o2}}{\partial V_{im}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial V_{on}}{\partial V_{i1}} & \frac{\partial V_{on}}{\partial V_{i2}} & \cdots & \frac{\partial V_{on}}{\partial V_{im}} \end{bmatrix}_{n \cdot m} \tag{2.11}$$

O Jacobiano pode ser usado no processo de minimização de erro entre posição atual e desejada numericamente, usando sua inversão J^{-1} e assumindo pequenas perturbações (δ). A Equação 2.12 mostra o relacionamento entre os sistemas de estado articular e cartesiano usando a inversão do Jacobiano Geométrico.

$$\delta q = J^{-1} \delta X \tag{2.12}$$

O problema com o Jacobiano é que frequentemente ele não é inversível devido a matriz não ser quadrada ou a pontos de singularidade causando problemas de guiagem e controle usando técnicas MGI analíticas. Métodos numéricos usando a pseudo-inversa de Moore-Penrose são uma alternativa para lidar com esses casos.

A pseudo-inversa de Moore-Penrose é um constructo matemático que permite a inversão de matrizes que não são quadradas ou em pontos de singularidade, e pode ser usada para minimização de erro, seguindo uma regra específica determinística, como a popular minimização em mínimos quadrados (BUSS, 2009). A Equação 2.13 e 2.14 são respectivamente usadas para o caso da matriz Jacobiano ter mais dimensão de coluna que linha, sendo "gordo", e no caso contrário, quando o Jacobiano é "alto", tendo mais dimensão de linha que de coluna.

$$J_i = J^T (J J^T)^{-1} \tag{2.13}$$

$$J_i = (J^T J)^{-1} J^T \tag{2.14}$$

Construindo um algoritmo para determinação de variáveis articulares, a Equação 2.12, se torna a Equação 2.15 usando a pseudo-inversa e notação explícita, e pode ser calculado iterativamente, gerando a Operação 2.16.

$$q_f = q_o + J_i(X_o) \cdot (X_f - X_o) \tag{2.15}$$

$$q_{i+1} \leftarrow q_i + J_i(X_i) \cdot (X_f - X_i) \tag{2.16}$$

Esse método é útil porque não requer o uso do complexo IGM analítico e permite a inversão de matrizes não-quadradas; mas pode requisitar mais poder computacional para realizar cálculos em tempo real. Sagliano e Theil (2013) e Jin et al. (2017) investigam métodos para melhorar a aplicação da pseudo-inversa do Jacobiano e sua performance de cálculo. Os métodos propostos são usados para resolver problemas de otimização complexos, como otimização topológica e geração de trajetórias aeroespaciais.

De posse do método de otimização agora é necessário tratar de políticas de geração de trajetória para levar o robô de um estado cartesiano corrente ao objetivo de forma suave.

2.5.4 Geração de Trajetória

A geração de trajetória é um subproblema do planejamento de movimento (NASCIMENTO et al., 2020), relacionado a determinar as referências de posição e orientação que o elemento terminal do robô, para o caso de manipuladores, deve seguir em seu trajeto até o estado objetivo. A configuração/disposição articular pode ser relevante. A trajetória pode ser descrita por curvas contínuas ou por pontos intermediários discretos conexos. A abordagem desse trabalho é baseada em pontos discretos e em uma política de trajeto ponto a ponto, sem políticas que favorecem transição para configurações específicas.

Existem diversas políticas e técnicas distintas para a geração de trajetória para que o robô siga uma sequência de estados. Cada uma coloca ênfase num aspecto distinto: velocidade de cálculo, dinâmica e controle. Elas podem levar em consideração outros fatores, como funções de estimativa internas baseadas em consumo de potência, fadiga, ou agentes externos e obstáculos (para evitar colisão). Marinho et al. (2019) descreve um *framework* que é altamente preocupada com dinâmica e controle para evitar colisões respeitando o ambiente e outras entidades. Em contraste, nesse trabalho o planejador de movimento (gerador de trajetória) é um simples filtro de dados, deixando percepção, planejamento de ação e controle para outros módulos especializados.

Ferreira (2011) usa *B-splines* para conectar pontos no espaço, garantindo que a trajetória irá passar pelos pontos desejados, enquanto evita variações abruptas em velocidade e aceleração. Esse método dá ênfase na performance dinâmica, mas requer a concatenação de múltiplos estados objetivo para justificar seu uso.

Nesse trabalho cada estado objetivo é tratado de forma independente, seguinda uma simples fila FIFO, sem preocupação direta com derivadas. O referenciamento de velocidade e aceleração não é explícito, ficando a cargo da dinâmica e do controlador robótico. Essa prática não é ideal para aplicações sensíveis, por razões de suavidade e comportamento previsível, mas é suficiente para a solução do problema exemplo explorado nesse trabalho.

Foram selecionadas técnicas que valorizam a simplicidade para geração de trajetória, pois são genéricas em aplicação, simples de entender, e são simples de se adaptar para ganhos de eficiência computacional em casos particulares. A política de trajetória mais simples é uma transição linear. Acoplado a ela são usados critérios de granularidade e distância entre-pontos constante. Os detalhes de cálculo e parâmetros na geração de trajetória são detalhados na Secção 3.5.3 e exemplificados na Secção 4.2.4.

3 DESENVOLVIMENTO DE ARQUITETURA E FERRAMENTAS

A arquitetura proposta busca aumentar a flexibilidade no suporte a domínios e especificação de algoritmos de processamento, através da capacidade de receber *snapshots* menos restritos e da importação de planejadores simbólicos para auxiliar na atuação em domínios reais, em tempo de execução.

Essa secção é dividida entre especificação teórica e genérica de ferramentas e arquitetura. O *design* genérico é importante para facilitar o transporte das soluções desenvolvidas para domínios laterais, tendo em vista que na aplicação dos modelos genéricos os módulos são implementados de forma a satisfazer o domínio de análise.

As ferramentas desenvolvidas para suporte a arquitetura são a PRD e o PSS. A primeira serve para representar informação na forma de predicados em objetos distribuídos no domínio, para adquirir informação e gerar *snapshots*. A segunda é um modelo de discretização do espaço de trabalho em pontos lógicos, para facilitar a implementação do domínio de planejamento e estruturar os processos de identificação e verificação.

A implementação com PRD e PSS é dividida em três módulos principais, Percepção RFID, Planejamento e Execução (Figura 6). Nesse procedimento é primeiramente necessário identificar a configuração e objetivo dos objetos, através de uma busca na PSS adquirindo dados na PRD dos objetos. A segunda etapa é alimentar um planejador, que faz uso de um planejador simbólico e um planejador numérico. Na terceira etapa acontece o retorno de um plano de ação para o módulo de execução, onde o sistema é atuado, alterando os estados das entidades. Essas ações devem se repetir até que o vetor objetivo de todos os objetos seja satisfeito com sucesso, auto ajustando caso necessário.

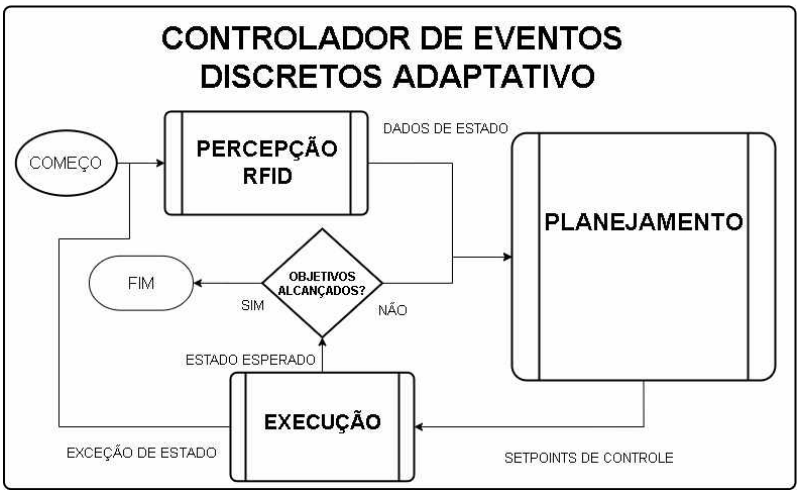

Figura 6 – Fluxograma de Controlador Adaptativo com Planejamento Automático

3.1 ESPECIFICAÇÃO DE ESTRUTURA PRD

A linguagem de marcação de dados (GIL; RATNAKAR, 2002) usada na PRD não está completamente definida. Após avaliação de linguagens de marcação populares, JSON (*JavaScript Object Notation*) é considerada inadequada pois opera em pares atributo/valor. XML (*eXtensible Markup Language*) é verbosa e não dá suporte simples para vetores e argumentos. Neste capítulo não será tratada a linguagem de marcação adotada, devido à especificação ainda indefinida, o que será abordado num trabalho futuro. Serão tratadas a estrutura PRD e dinâmica de marcadores e predicados.

A PRD e dados auxiliares são representados usando marcadores, com notação *:marcador*. Logo a descrição de um estado atual pode ser inferida buscando por um marcador *:current* e lendo a informação entre chaves, usualmente escrita como predicados separados por vírgulas. Uma série de objetivos consecutivos pode ser localizada usando uma notação *:objective-N*, onde *N* referencia o enésimo objetivo. Informação auxiliar pode ser tratada usando o mesmo sistema, como exemplo *:temperature* e *:priority* são marcadores que podem ser usados para suplementar ou confrontar informação com relação aos dados históricos e desejados para o objeto.

A Figura 7 apresenta um exemplo de representação PRD para mostrar como estados atual e objetivos são encontrados. Em *:current* três predicados distintos são introduzidos para descrever a situação corrente, referenciando outros objetos e entidades do domínio, enquanto *:objective-O* dois predicados são usados para descrever o estado desejado, com status() referenciando a completude de um estado. O marcador *:priority* pode ser usado para indicar preferência, caso haja discordância de objetivos entre agentes ou recursos limitados. É a tarefa do agente ativo processar e inferir informação com os predicados coletados.

```
...
:current {
  at(PathID(1)),
  dev(Application(4)),
  conjugate(PartType(2) | PartType(7)) }
:objective-0 {
  at(PathID(3)),
  status(Processed) }
:priority { 2 }
...
```

Figura 7 – Estrutura de dados PRD em agente passivo

PRD é uma abordagem utilizada para guardar, ler e escrever predicados em cartões RFID, efetivamente dando poder decisório para agentes passivos. Imagine a peça marcada

P numa linha de produção, até que ela chega numa divergência, conforme retratado na Figura 8, onde o agente ativo *robô R* precisa decidir qual caminho a *peça P* irá tomar, *A* ou *B*. Não há informação externa ou geométrica para influenciar a decisão, e qualquer agente ativo deveria atuar da mesma forma.

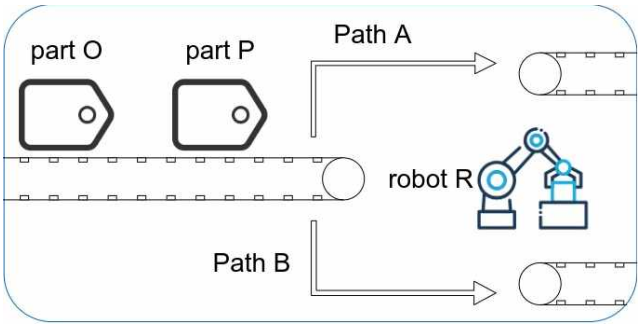

Figura 8 – Bifurcação em linha produtiva

Essa decisão pode ser tomada se o próprio agente passivo fornecer a informação ao agente ativo, para que ele possa decidir e agir. Para descrever a aplicação da PRD nesse exemplo, o *robô R* pode conseguir informação sobre qual caminho a *peça P* deve seguir, seja diretamente relacionado ao caminho em si, ou pelo destino, como ir para a secção de embalamento ou para o controle de qualidade. Essa informação pode ser acessada no marcador *:objective-0*, conforme a Figura 9.

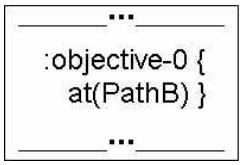

Figura 9 – Marcador PRD *:objective-0* dentro de *part P*

Nesse caso a *peça P* tem armazenado explicitamente que o caminho *B* é um objetivo. Isso pressupõe que algum aplicativo seja capaz de atribuir e gravar essa informação na *tag* em algum momento antes da tomada de decisão. *Robô R* irá processar esse marcador e inferir, usando uma base de regras e dados, que deve agir de forma a levar a *peça P* até o caminho *B*. Esse método é mais poderoso que usar outras fontes de dados como a PNRD/iPNRD, pois a representação em predicado agrega flexibilidade como uma linguagem com mais poder de notação prático e aproxima a grafia entre a estrutura de dados do objeto e do planejador automático; todavia, é um formato muito distinto das linguagens de programação do chão-de-fábrica. Conforme *R* movimenta qualquer *peça X*, ele pode atualizar os dados PRD dentro do cartão de cada uma.

Dados individuais PRD podem ser agrupados e usados para produzir descrições mais ricas e aplicações mais complexas. Para observar o acoplamento de efeitos em dados

agrupados, imagine que a *parte O*, na Figura 8, tem um predicado que especifica que ela deve seguir o mesmo trajeto que a *parte P* (Figura 10). Esse objetivo é uma referência implícita ao estado que será alcançado pela *parte O*. Isso causa informação em P e na decisão sobre P para também afetar O, agregando dependências em causalidade, portanto complexidade.

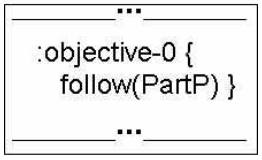

Figura 10 – Marcador PRD *:objective-0* dentro de *part O*

Outra característica da PRD é sua permissividade para definição de múltiplos objetivos, seja na forma de agregação "*AND*", condições que devem ser satisfeitas simultaneamente, "*OR*", condições que basta satisfazer uma do agrupamento, e "*NOT*", negação de uma condição. Outras agregações são deriváveis, dando suporte para geração de descrições de objetivos mais flexíveis.

3.2 ESPECIFICAÇÃO DE ESTRUTURA PSS

O PSS é uma estrutura de dados, normalmente representada por uma ferramenta gráfica, como dígrafos e Redes de Petri, usada para simbolizar posições discretas e relações de adjacência no escopo do domínio físico, útil para discretizar o movimento contínuo em transições discretas mais simples. Essa ferramenta faz um mapeamento biunívoco entre regiões do espaço e símbolos, podendo ser utilizada para gerar uma descrição do domínio do sistema e para problemas de locomoção.

Exploração e busca no PSS podem ser feitas usando diferentes técnicas e com níveis distintos de complexidade. Exploração usando máquinas de estado é a forma mais simples de execução e requer esforço computacional mínimo, com complexidade temporal e espacial $O(n)$, pois basta seguir um vetor pré-estabelecido de pontos, enquanto outras técnicas com busca online (em tempo de execução) tem custo de busca acoplado (KORF; REID, 1998). Métodos offline (fora do tempo de execução) de otimização podem incluir condicionais, conforme discutido em Souza, Brito e Tavares (2019).

A Figura 11 mostra um espaço de estados 3x3 representado por uma Rede de Petri. Nessa representação são dados os estados e transições possíveis num espaço físico com transições bidimensionais. O *token* (marcação), representa o estado inicial. Nesse espaço pode acontecer movimentações que seguem uma ordem pré-estabelecida, como numa máquina de estados, ou movimentações com base em lógica ou ainda movimentação

com base em busca online. Nesse espaço de estados limitado a implementação mais simples e barata computacionalmente é em máquina de estados finita.

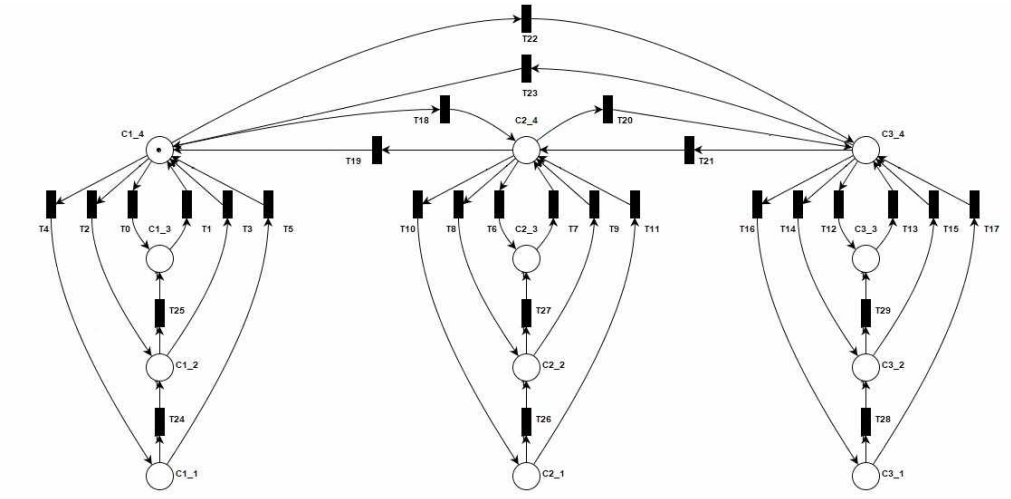

Figura 11 – Uma estrutura PSS 3x3 representada por uma Rede de Petri

No escopo de planejamento, o PSS é usado como auxiliar em momento de geração das descrições de domínio e problema em PDDL. Isso faz a ponte entre a interação com o domínio real e o planejamento abstrato.

3.3 INTEGRAÇÃO DE PLANEJAMENTO, PRD E PSS

De forma a gerar um modelo de planejamento automático simples e realizável, o modelo do sistema deve ser do tipo discreto, e o ambiente deve ser subjugado a possibilidade de distúrbios.

Quanto ao PSS, planejamento de trajetória online é o ápice de eficiência de movimentação mecânica, mas a um custo computacional superior a métodos mais simples, pois exige uma busca no espaço de movimentação. Na prática, a dimensão do PSS é pré-estabelecida e o agente ativo conhece sua própria posição e limites de alcance (dado o modelo espacial e as restrições cinemáticas e dinâmicas).

Enquanto o planejador é responsável pela tomada de decisão em modelos a eventos discretos, o controlador deve ter uma interface para sensoriamento e ação, o que pode incluir uma interface com o *driver* do atuador. De forma a estruturar o sistema adaptativo, a abordagem PRD propõe que sensores sejam cartões RFID nos agentes passivos, e leitores na função de leitura nos agentes ativos enquanto atuadores são agentes ativos (como um robô manipulador) acoplado do leitor RFID na função de escrita. Isso estabelece que a interface entre atores e agentes passivos é feita pela RFID usando a estrutura PRD.

O domínio físico adiciona restrições e complexidade que não estão presentes em domínios puramente computacionais e idealizados, essas características devem ser tomadas

em consideração para a estrutura da arquitetura PRD e modelos de funcionamento e domínio. Algumas das preocupações emergentes são: configurações de *snapshot*, na forma de identificação do estado do conjunto de agente passivos, bem como inferir se o estado objetivo coletivo é alcançável ou não; dinâmica de movimento e controle, na forma de usar o nível adequado de abstração para planejamento e para execução; e observabilidade parcial, para compreender que a visão de domínio atual pode estar errada devido a perturbações não percebidas. Mesmo que o robô manipulador rode sob um sistema parcialmente observável, devido ao fato de que cada bloco é localizado e lido em momentos diferentes, exceções podem ainda ser observadas, quando o sistema interage explicitamente com um objeto ou relação não-conformante, por exemplo um mesmo item lido múltiplas vezes.

A arquitetura do planejador automático integrado com PRD e PSS é apresentada na Figura 12, sendo outra forma de visualização do ciclo descrito na Figura 6. Nessa arquitetura o planejador é alimentado com dados advindos dos processos de varredura, identificação e inferência na PSS com PRD e de uma descrição em PDDL do domínio. O controlador executa uma série de passos e compara com a expectativa de estado a ser observada. No sistema há ainda a ação de ler a PRD de agentes e sobrescrever a PRD atualizando os dados, usando o leitor RFID. O sistema deve ser retroalimentado para autocorreção.

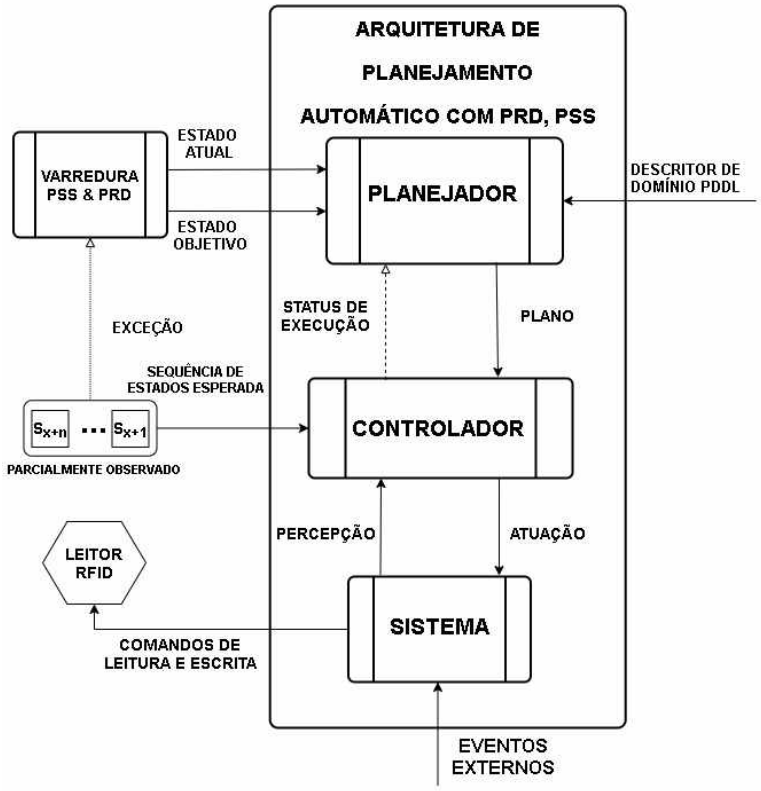

Figura 12 – PRD e PSS unidos ao Planejamento Automático para Controle a Eventos Discretos com capacidade de adaptação devido à retroalimentação da percepção em tempo de execução

O módulo de Percepção RFID observa o ambiente, através da captura de dados PRD dos agentes passivos (Identidade, Estado atual e Objetivos) movendo o agente ativo integrado com um leitor RFID embarcado em cada posição discreta de interesse, seguindo o PSS. Esse passo é terminado apenas quando o PSS é exaurido.

O próximo passo é de percepção, a inferência sobre os dados PRD agrupados e objetivos, chamada de Inferência de Estado. Inferência de estado atual também pode ser definida pela informação coletada de posicionamento no PSS. Essa informação pode ser confrontada com a etiqueta *:current*, para observar inconsistências. Estado objetivo é gerado agrupando todos os predicados *:objective-O*. Também deve ser verificado se esse estado é válido (realizável e consistente).

Identificação de exceções é um problema complexo, devido a distintas situações, como muitas configurações serem impossíveis de se realizar fisicamente, objetivos em oposição, ausência de objetos referenciados e outros. A aplicação de regras gerais para detecção rápida de exceções pode melhorar a reação do sistema; porém uma solução geral ainda é um desafio. Se uma exceção foi observada, o processo de identificação da situação corrente deve ser reiniciado e um supervisor notificado. Caso contrário, tudo ocorra corretamente e sem problemas, as observações são transferidas como entrada (Dados de Estado) para o módulo de Planejamento. A Figura 13 detalha o módulo de Percepção RFID.

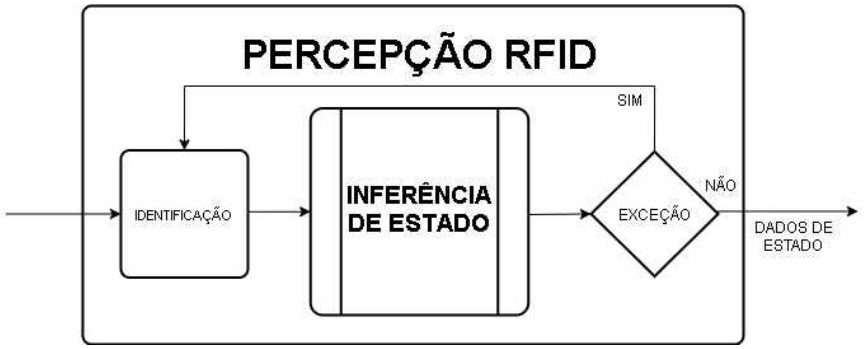

Figura 13 – Módulo de Percepção RFID da PRD

O módulo de planejamento é um planejador de dois níveis, responsável por ambos o planejamento de estados discretos quanto pelo planejamento de movimento físico. A Figura 14 mostra que ambos sub planejadores precisam de uma base de conhecimento, que é composta de uma descrição do sistema acoplada de restrições.

O planejador discreto requer dois arquivos distintos para operar, um descrevendo o domínio e outro o problema. Uma descrição básica do domínio é composta da declaração das classes existentes e ações existentes como funções com precondições e consequências. Uma descrição de problema básica inclui um repositório de objetos existentes, *snapshot* atual e condições-objetivo, capturadas na coleta de dados PRD.

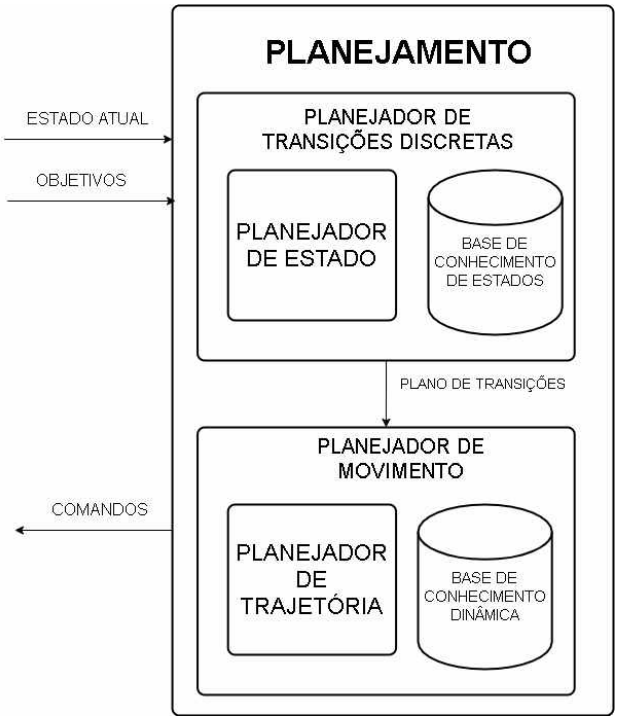

Figura 14 – Módulo de Planejamento

O planejador contínuo requer o vetor completo de ações do planejador discreto, uma descrição dinâmica, como o DGM para robôs manipuladores, e o estado atual do atuador como entrada. O planejador deve contar ainda com política de geração de trajetória e otimização, além de uma definição dos algoritmos internos e condições de parada. A saída do planejador de movimento é uma corrente de vetores-solução, os *setpoints* de entrada do bloco executor.

O módulo de execução é composto pelo controlador do agente ativo junto de uma base de conhecimento intérprete, que define a sequência de execução e controle. Em seguida há uma inspeção, similar à Percepção RFID, para checar se o estado alcançado é o desejado. Caso os objetivos tenham sido corretamente alcançados, o leitor do robô irá atualizar os dados PRD dentro de cada tag (Figura 15). Caso um ciclo encerre corretamente, uma avaliação de conclusão de tarefas é realizada para iniciar o próximo ciclo ou finalizar o processo, conforme apresentado na Figura 6.

3.4 PLANEJAMENTO COMO SERVIÇO

A Figura 16 retrata a visão topológica do sistema, dividido em camada física e camada virtual, onde a camada física representa o nó local, o controlador do robô localizado no ambiente de interação e a camada virtual representa o nó virtual, um servidor remoto ou local para processamento de dados.

Na camada física há a interação com o domínio real, onde ocorre a percepção e

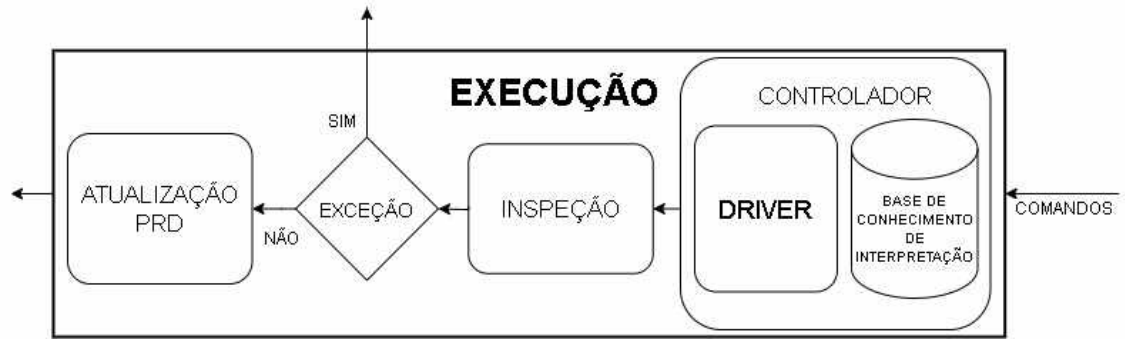

Figura 15 – Módulo de Execução e Inspeção

ação, e perturbações advindas de agentes externos podem ocorrer sobre os objetos. A camada virtual é responsável pela maioria do processamento dos dados, de forma que há vantagem significativa no uso de uma máquina-servidor para processamento, tendo em vista que o processador do controlador físico é limitado. O fluxo básico do processo é de percepção para planejamento, para atuação, onde cada processo conta com informação do domínio para funcionar corretamente. No planejamento, em especial, é dada ênfase na relevância dessa base de conhecimento, já que esse processo está localizado num dispositivo que não manipula diretamente o domínio.

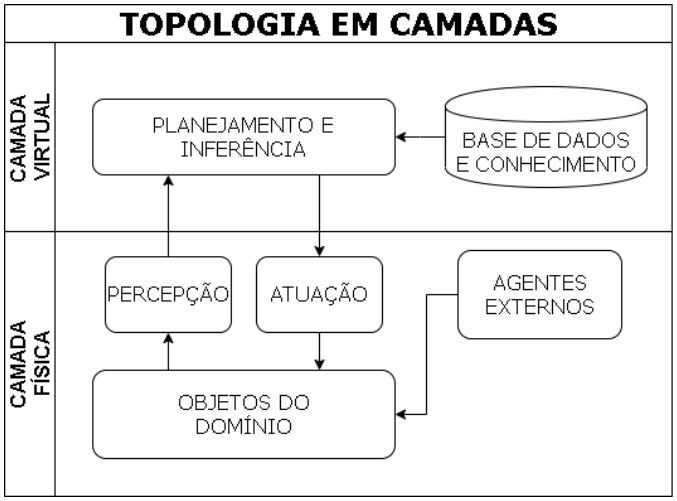

Figura 16 – Topologia funcional do sistema proposto

Para ganhos de performance, o cliente pode requisitar o serviço, ele pode acessar um servidor na borda da rede ou na nuvem, desde que a rede conectiva esteja em condições úteis. Uma visão prática da divisão de processo e fluxo de processamento é dada nas Figuras 17 e 18. A divisão de responsabilidade do sistema é dada na Figura 17, onde o cliente, o processador robótico, é responsável por interagir com o sistema físico e de preparar os dados para planejamento, e o servidor por retornar um sequência de referências de controle, portanto a aplicação é dividida entre os dois nós, de forma que a percepção e atuação estão localizadas no cliente e o planejamento no servidor. Na Figura 18 fica claro

o padrão de fluxo de processamento, que é totalmente sequencial e causal, sendo o serviço invocado sempre que for necessário planejar ou replanejar devido a identificação de uma exceção. Devido à natureza do processo a operação é *half-duplex* na prática.

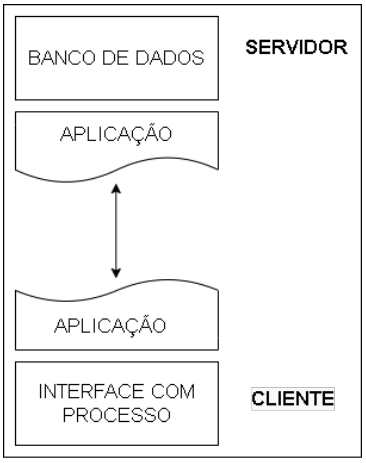

Figura 17 – Divisão de responsabilidades entre cliente e servidor

A forma de troca de dados na Figura 18 é por pacotes de texto completos, sendo essencial garantias de entrega ordenada, logo o uso de protocolos orientados à conexão é favorecido (COULORIS; DOLLIMORE; KINDBERG, 2007). Pelo desenho arquitetural é possível que um servidor trate múltiplos clientes, todavia essa funcionalidade é um trabalho futuro.

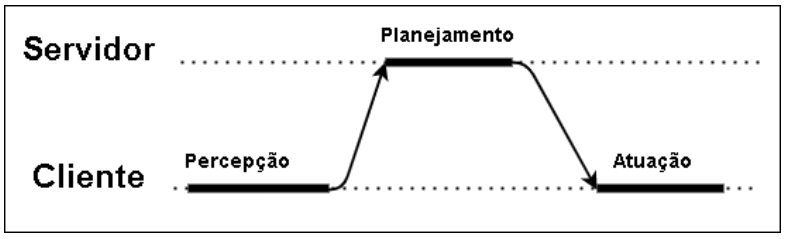

Figura 18 – *Pattern* de fluxo de processamento

A aplicação deve acontecer de forma independente das camadas de base do protocolo OSI (BOCHMANN, 1990), para fins de transparência e portabilidade. No acesso ao serviço de planejamento informações básicas de domínio, problema, modelos dinâmicos e estado geométrico atual do agente devem ser repassadas (ou assumidos com base num banco de dados ou comportamento pré determinado) para que o servidor possa retornar comandos no formato apropriado. Detalhes de implementação são tratados na Secção 4.1.

3.5 DETALHES E INTEGRAÇÃO DE PERCEPÇÃO, PLANEJAMENTO E CONTROLE

Para integrar planejamento e ação de forma apropriada é necessário a geração de um plano com nível suficiente de refinamento da abstração para que a máquina possa interpretar corretamente os passos (GHALLAB; NAU; TRAVERSO, 2016). Isso identifica que os passos de alta abstração necessitam de detalhamento, gerando sub passos (filhos), onde esses detalhes nem sempre podem ser embarcados em seus procedimentos pais como constantes, pois normalmente são variáveis com a situação, sendo necessário para sua definição métodos matemáticos ou sub planejamento.

Como exemplo, imagine um robô móvel de propósito geral. Após receber uma tarefa de alto nível, essa tarefa deve ser refinada em passos mais simples, iterando até que sejam concretas o suficiente para atuação. A Figura 19 exemplifica os níveis de abstração na deliberação, como um simples comando "traga o objeto *o7* até o local *room2*" é muito complexo para ser diretamente tornado ação, requisitando refinamento progressivo para poder agir.

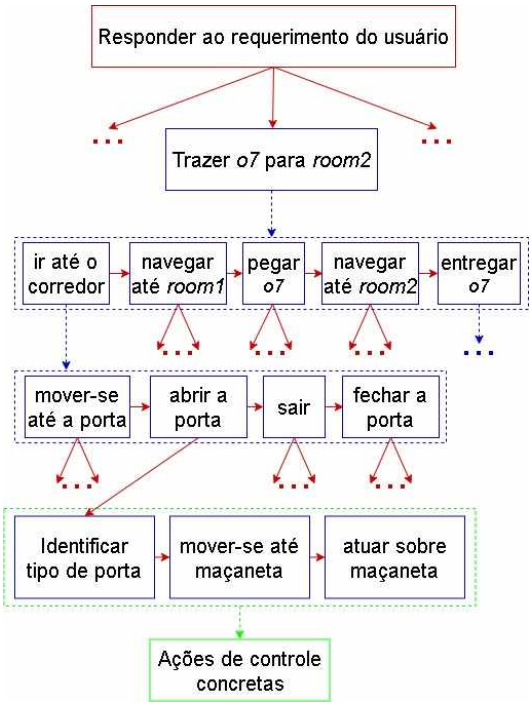

Figura 19 – Múltiplos níveis de abstração na deliberação - adaptado de Ghallab, Nau e Traverso (2016)

Para evitar trabalhar com planejamento automático em múltiplos níveis de abstração, a abordagem PRD proposta para implementação é definida com dois níveis de planejamento. O primeiro é usado para gerar passos discretos e o segundo nível é usado para converter o passo em entradas de posição, por meio de métodos matemáticos, para ser consumida pelo executor na forma de *setpoints* de controle.

3.5.1 Percepção

No bloco prévio ao planejamento, o bloco de percepção, é onde os dados capturados do processo de varredura são tratados para gerar uma descrição do estado corrente e objetivos e alimentar o planejador discreto. É importante que a formatação de saída esteja adequada a entrada para o bloco de planejamento.

Após localizar e ler os dados PRD dentro de cada etiqueta, os predicados sobre estado corrente e objetivos são unidos para gerar a especificação do problema, conforme o processo descrito na Figura 20. Pode-se observar que há duas fontes de informação advindas da identificação, as PRD (lógicas) e PSS (físicas). Dentro da máquina de inferência os dados são organizados de forma a alimentar o processo de inferência. Esse processo é responsável por gerar o problema PDDL, e logo em seguida checar se não houve exceção (discutidas na Secção 4.1.1). A descrição do domínio deve, também, ser enviada/referenciada ao planejador como saída, para casos não pré-estruturados. Na percepção é possível o uso de multiagentes colaborativos para a identificação e inferência.

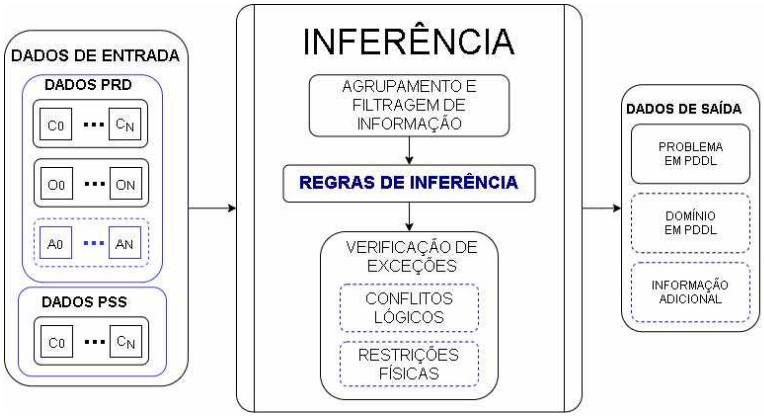

Figura 20 – Integração de dados e inferência PRD

A máquina de inferência pode ser implementada de duas formas distintas, conforme as Figuras 21 e 22. A Figura 21 descreve o processo de inferência usando um compilador da linguagem de marcação de dados PRD para PDDL, onde é necessário alimentar o compilador com informação externa sobre o domínio e combinação lógica dos marcadores e predicados para que os processos de *merge* de dados e de regras e exceções sejam executados. Aho et al. (2006) traz uma introdução a arquitetura e construção de compiladores, onde os processos de *tokenização* e *parse* individuais correspondem aos procedimentos de análise léxica e sintática para cada objeto. Os procedimentos seguintes são relacionados a fusão de dados, para identificação completa dos objetos de domínio percebidos, a geração de PDDL e análise de exceções.

No software *ItSimple* (VAQUERO, 2007) foram construídos tradutores usando as linguagens Java e Python para traduzir PDDL dos modelos gráficos, representados computacionalmente em XML, e para compilar/decompilar para PROLOG. Devido a

natureza de microcontroladores *low profile*, isto é, de baixos recursos, o uso de linguagens que se baseiam em ambientes de execução virtuais é inviável. Algumas das possíveis linguagens para implementação de alta performance e baixo custo de memória são C, C++ e Rust. Essa implementação será desenvolvida num trabalho futuro.

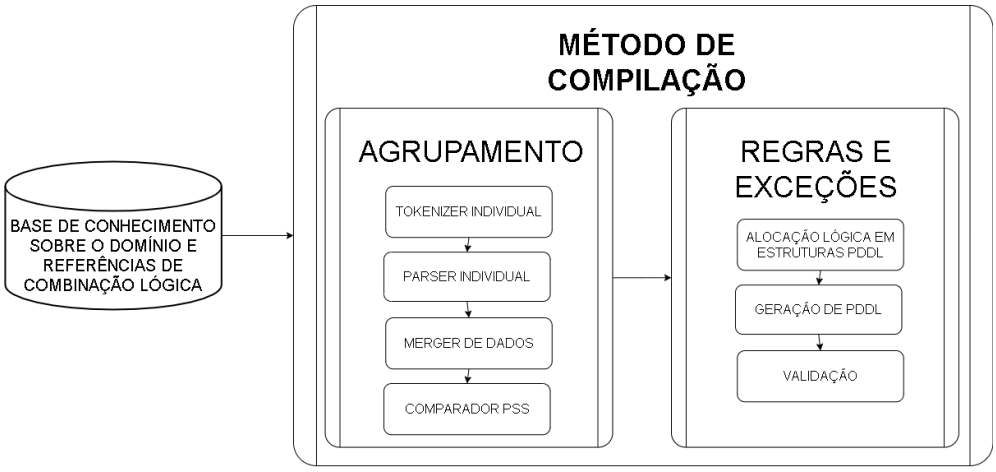

Figura 21 – Implementação de Máquina de Inferência por Compilação

No método pré-estruturado (Figura 22), os marcadores e predicados passíveis de uso são pré introduzidos, de forma que uma busca usando expressões regulares (ELLUL et al., 2004), é capaz de rapidamente encontrar os marcadores e predicados relevantes para leitura e *parse*. As regras de conversão em PDDL se tornam mais simples e é facilitado que sejam manualmente explícitas. Esse é o método adotado nesse trabalho.

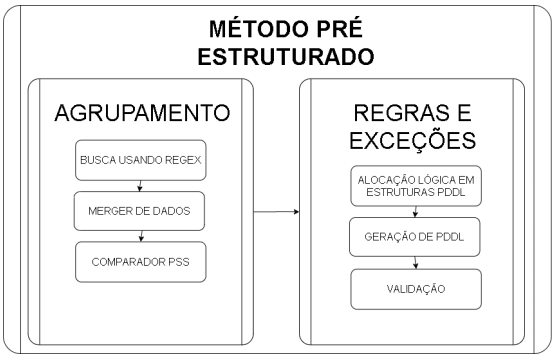

Figura 22 – Implementação de Máquina de Inferência pré-estruturada

3.5.2 Planejamento de Ação

O problema de planejamento discreto é caracterizado pela busca num espaço abstrato, de forma a levar o sistema de uma condição inicial até o alcance dos objetivos, pelas regras do domínio e objetos existentes. Essa etapa de planejamento dá suporte para um único ator e para multiatores colaborativos.

PDDL é usada nesse trabalho para alimentar o planejamento automatizado, na primeira fase de planejamento. Foi escolhida porque é uma linguagem em crescimento e evolução constante, usada no escopo da comunidade de planejamento automático, já que foi a linguagem oficial de diversas ocorrências da IPC (Competição de Planejamento Internacional).

Na Figura 23 é apresentado os dados de entrada e saída do planejador de estados, bem como seus parâmetros internos de funcionamento. Os dados de entrada devem contar, obrigatoriamente, com o problema identificado. O Domínio também deve ser alimentado, caso o planejador não o possua, e dados adicionais podem ser providos.

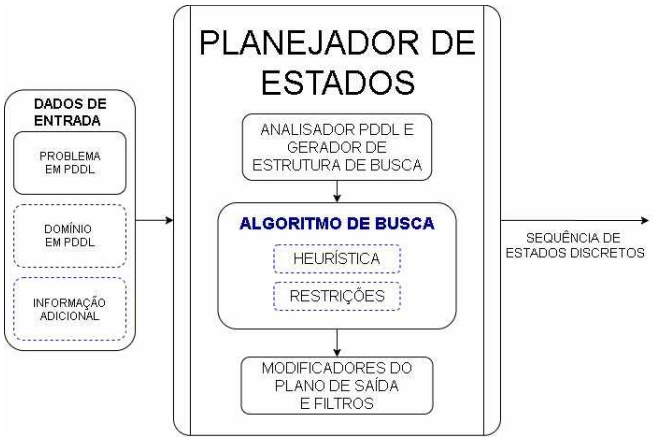

Figura 23 – Detalhes do Planejador de Estados

O planejador conta com um "tradutor" para interpretar os arquivos PDDL e montar o problema para busca. Depois um algoritmo de busca é utilizado, podendo contar com heurísticas e restrições. Por fim, modificadores e filtros são aplicados no plano de saída para adequação ao passo de planejamento de movimento.

3.5.3 Planejamento de Trajetória

O planejamento de trajetória se baseia no uso de métodos analíticos ou numéricos para geração de trajetória e *setpoints* de controle, minimizando erros numéricos e satisfazendo restrições impostas, conforme discutido na Secção 2.5. Essa etapa de planejamento deve ser realizada para cada ator.

Na figura 24, os dados de entrada, são alimentados iterativamente até que todas as transições discretas sejam convertidas em movimentação contínua para entrada no controle físico. Na Inicialização, o sistema supõe posse do DGM, política de trajetória e otimização, mas é possível que sejam passadas como entrada.

Os dois *loops* (ciclos) internos, de geração de trajetória e otimização são usados respectivamente para gerar a trajetória de movimentação e para reduzir o erro numérico

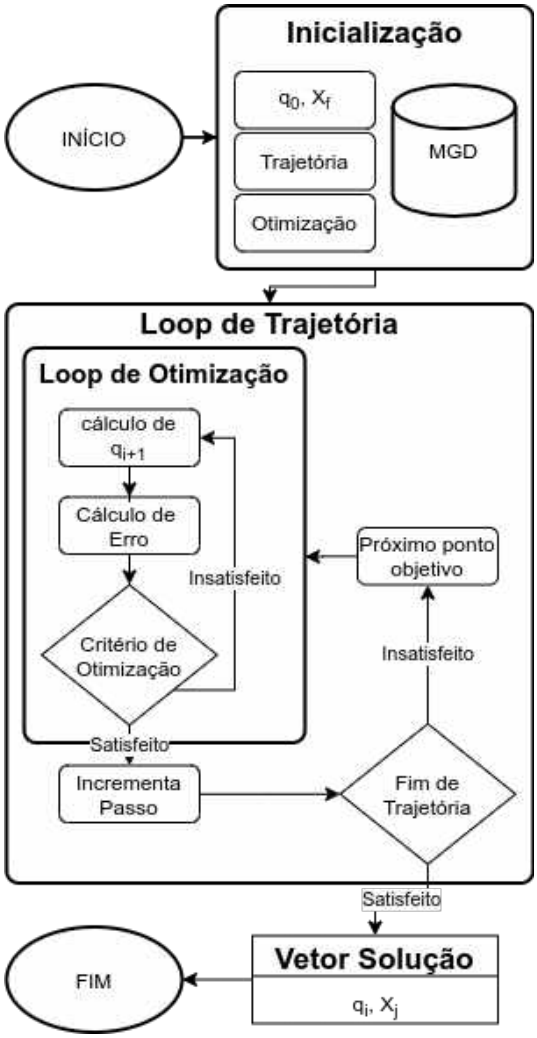

Figura 24 – Detalhes do Planejador de Movimento

na setagem do vetor de *setpoints* q_f. Os *loops* são continuados até que critérios de parada sejam satisfeitos, como número de iterações i_{max} ou norma do erro E_n.

O vetor solução final, após iterar sobre todas as entradas discretas do bloco de planejamento de movimento, é uma série ordenada de vetores de *setpoint* das articulações do robô. A sequência da série faz com que o robô passe por todos os estados discretos previamente estabelecidos. Basta agora, que o controlador físico interprete os dados de entrada e converta num problema de controle.

3.5.4 Interpretação e Controle

O bloco de controle da máquina é basicamente uma sequência de *setpoints* a serem alcançados em um problema de controle. O vetor de *setpoints* entra na máquina e é pré tratado internamente e alocado num *buffer*, no interpretador. O *driver* vem consumindo as referências do *buffer* até o término da execução, gerando um novo estado alcançado para o domínio (Figura 25).

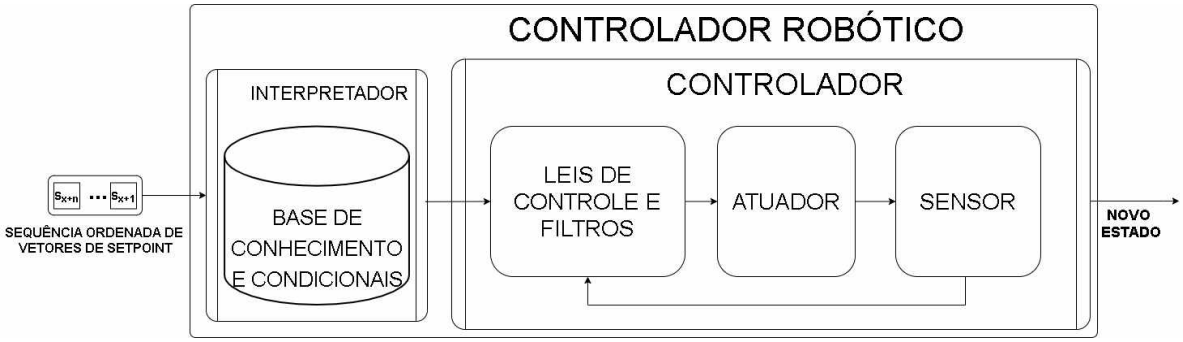

Figura 25 – Submódulos de Atuação do Controlador, do módulo de Execução

Os detalhes de controle, da seleção do algoritmo e método de projeto, são altamente acoplados ao robô e condições específicas. Porém algumas características comuns podem ser observadas, tais como prevalência da presença de múltiplas entradas e saídas, acoplamento leve a severo, uso de atuadores instrumentados eletrônicos ou hidráulicos/pneumáticos e uso de controladores digitais (LYNCH; PARK, 2017). Logo, especula-se que trata-se de uma planta MIMO (Múltiplas Entrada Múltiplas Saídas), microcontrolada por uma máquina digital, atuada por dispositivo analógico e instrumentada com sensor eletrônico (assumido como de posição/ângulo). A relevância dessa pré análise está relacionada ao que esperar do sistema e na construção da arquitetura em si, tendo em vista que a arquitetura deve ser genérica e robusta o suficiente para suportar a mudança de robô ou de algoritmo de controle.

Em Kelly, Davila e Loria (2005) discussões, modelos e técnicas específicas para o controle de robôs são levantadas, em especial as que são tradicionais na indústria. Para o controle de posição e movimento, há a prevalência de técnicas de controle desacopladas, baseadas em controle PID (Proporcional Integral Derivativo). Controladores mais sofisticados, com características adaptativas ou de aprendizado, ou mesmo MIMO com custo computacional superior são evitadas quando possível.

No projeto analítico de controle é necessário o modelo dinâmico da planta, além do método de cálculo, atuação e sensoriamento (FADALI; VISIOLI, 2013). Essas especificidades não são abordadas nesse trabalho, porém alguns requisitos de controle podem ser levantados, dado o problema típico de controle de manipuladores robóticos (ZHANG et al., 2013). Requisitos ideais de controle, para a aplicação:

- Erro nulo em regime permanente;
- Sobre-sinal nulo;
- Tempo de acomodação mínimo.

Ainda, sobre outras características de performance e comportamento:

- Magnitude de oscilações nula;
- Desacoplamento total entre saídas;
- Imunidade a perturbações.

Claramente essas são condições idealizadas, desejáveis na maioria dos processos. A determinação de valores numéricos para os requisitos ou tradução em termos de parâmetros de projeto de controle como polos, zeros, margens, taxa de amostragem, etc, é dependente da aplicação especificada.

As técnicas de controle mais comuns empregadas na indústria para a finalidade de controle de robôs são baseadas na família PID, usando um controlador digital. Contudo, novas técnicas baseadas no consumo de dados, aprendizado ou controles com custo computacional superior são agora possíveis (YURISH, 2018). Logo, um estudo sobre escolha da técnica deve ser feito, com base nas características do robô, domínio e recursos. No exemplo explorado na Secção 4.2, a métrica de controle é justificada.

4 EXEMPLO E RESULTADOS

Nessa secção são apresentadas as formas de implementação de percepção e planejamento, seguida pela modelagem de um domínio, exemplos e análises.

4.1 IMPLEMENTAÇÕES

4.1.1 Implementação da Percepção

Na versão corrente a percepção é emulada virtualmente identificando dados na estrutura PRD em arquivos num diretório, pelo método pré-estruturado, usando C. A união da informação relevante, no modelo GISP, permite que os dados sejam analisados e traduzidos para uma especificação de problema PDDL.

É usada informação da definição do domínio e do PSS para pré-estruturar informação do problema. Os dados são buscados no predicado ID e em marcadores *:current* e *:objective-0*, onde os predicados internos ao marcador são recolhidos e alocados numa estrutura que representa cada objeto, sendo respectivamente a identidade, estado corrente e objetivos do objeto. Da união dos objetos identificados no diretório são feitos processos de análise para determinar os predicados para popular o cabeçalho, *:init* e *:goal* do arquivo problema PDDL. A validação do problema gerado, através de uma rotina automatizada, é um trabalho futuro.

Na versão corrente a quantidade de blocos e atores no domínio é dinâmica, isto é, a quantidade de arquivos com informação PRD dos blocos e atores não é pré-estrutura. Contudo, o PSS possui tamanho e predicados pré definidos (dimensão 4x4). Outras limitações da versão corrente são apenas um objetivo por arquivo e é suportado como objetivo apenas os predicados ligados a posicionamento direto (*holdblock* e *holdarm*). Os predicados estão alocados internamente no código, como *hardcode*, o que torna a solução mais específica, porém numa implementação para microcontroladores esse também seria o caso.

A árvore do projeto é composta por quatro diretórios alinhados no mesmo nível. No primeiro diretório, o diretório *source*, há os arquivos fonte, no diretório *prebuilt* é dada as pré-definições do PSS e dos atores, no diretório *domain* há os arquivos *.in que emulam os blocos com suas etiquetas RFID no domínio, e o diretório final *output* contém as saídas do processo de inferência. O processo de identificação e inferência é invocado com um argumento de nome do arquivo de saída. Mais detalhes são dados na Secção 4.2.5.2.

4.1.2 Implementação do Planejamento de Ação

O planejador automático de escolha é o *Fast-Downward*, um planejador *open source* (de código aberto) completo, mantido por grupos de pesquisa (SEIPP; ROGER, 2018), desenvolvido principalmente em Python e C++, que usa a licença *copyleft* GPL (SABINO, 2011), o que o torna útil em ambientes acadêmicos. Ele suporta diversas linguagens de entrada, algoritmos de análise semântica, busca e heurísticas, e retorna informação detalhada sobre processamento, procura e, claro, o plano gerado.

Foram realizadas três implementações distintas com meio de transmissão de dados diferentes, todas em sistema operacional Linux. O fluxo dos processos é o mesmo: criação da interface; acesso do cliente e envio do problema; processamento do plano; retorno de solução. A chamada de processos e filtragem de *strings* é feita usando bash. As implementações são:

1. Acesso Serial;
2. Acesso via http por página WEB;
3. Acesso via *socket* em servidor remoto.

Na primeira abordagem o cliente é conectado por cabo ao servidor, sendo uma alternativa *Edge*. É utilizada uma porta \dev\ttyACM* para conexão USB programada na linguagem C usando as abstrações POSIX. Quando o cliente é conectado à máquina, o sistema operacional identifica o *device driver* adequado e atribui dinamicamente o endereço de acesso. O servidor deve ter conhecimento do endereço correto para troca de dados. O cliente envia o problema PDDL e recebe o plano de ação para consumo.

Na segunda abordagem foi usado um WEB serviço em Apache2, programado em PHP, para simular o acesso pela Internet no *localhost* (127.0.0.1:80), fazendo com que a solução seja visível numa página WEB. Os arquivos do projeto são tidos como de propriedade do usuário e grupo "*www-data*", além de permissão de execução, escrita e leitura universal para que possam ser acessados externamente. Os dados do problema PDDL são enviados como argumento de um atributo por conexão remota ou por entrada manual via *browser*. Esse método pode ser adaptado para uso *Edge* ou *Cloud*.

Na terceira abordagem é utilizado um servidor remoto AWS EC2© para acessar o serviço de planejamento, sendo uma solução *Cloud*. O acesso é feito usando a API *socket* POSIX, que tem como infraestrutura os protocolos TCP/IP (CONNER, 2015) para essa aplicação, sendo percebido por meio de um simples processo *monothread*. A máquina-servidor possui permissões para uma faixa de IP e *ports* para que clientes se conectem. O cliente deve possuir informação do IP da máquina-servidor e o *port* do serviço. O processo servidor faz a conexão com o *socket* e entra em estado de escuta, aguardando que o cliente se conecte. Quando o cliente solicita conexão ele é aceito com base em seu IP

de origem. O cliente envia o problema PDDL e o plano de ação é retornado para consumo pelo cliente.

4.1.3 Implementação do Planejamento de Trajetória

O planejador de movimento de escolha foi desenvolvido para esse trabalho, seguindo as especificações da secção 3.5.3. O planejador foi primeiramente desenvolvido em MATLAB©, com a intenção de ser futuramente redesenhado para C++, para ganhos de performance, portabilidade e distribuição livre (licença MIT), além de facilitar a cooperação com outros processos e uso em tempo real.

A implementação do planejador segue a descrição apresentada na Figura 24. O DGM é uma informação modelada fora do tempo de execução, estando acoplada a execução como um *script*/função separado. Condição inicial e objetivos são informações inicializadas no topo do fluxo do processo, logo depois são inicializados os parâmetros de otimização e trajetória e por fim o sistema realiza um loop triplo com o propósito de calcular todas as soluções concatenadas de coordenadas articulares. Funções nativas do MATLAB©, como a função de cálculo dos ângulos de Euler do elemento terminal e a pseudo-inversa são utilizadas.

4.2 ESTUDO DE CASO: MUNDO DE BLOCOS

O Mundo de Blocos é um domínio de planejamento conhecido na área de inteligência artificial (GUPTA; NAU, 1992). O objetivo é mover blocos de uma situação inicial até um estado global desejado. O planejamento no Mundo de Blocos foi demonstrado como um problema difícil para encontrar um plano ótimo, já que é *NP-hard* no pior caso (GUPTA; NAU, 1992). Encontrar uma rota ótima significa mudar o estado dos blocos para o objetivo com um custo mínimo, onde o custo é geralmente a soma total de todas as transições disparadas. Neste trabalho, o custo é definido no senso comum, com custo unitário para todas as transições/ações.

Essa secção apresenta a modelagem de domínio, junto da implementação e emulação de solução para uma versão de Mundo de Blocos mono e multi ator.

4.2.1 Especificações do Mundo de Blocos e Modelagem

As regras para o mundo de blocos proposto seguem o padrão da literatura, apenas com alguns ajustes para garantir com que se comporte corretamente no mundo físico. A Tabela 1 introduz o conjunto de regras em linguagem natural. Nota-se a preocupação em evitar colisões e comportamento que não está de acordo com as regras do espaço, tendo em vista que o plano de trabalho inclui a vertical.

Tabela 1 – Especificação em regras do Mundo de Blocos abordado

Regras	Descrição
1	Nenhuma entidade pode ocupar a mesma posição, salvo um ator e um bloco
2	Apenas um bloco pode ser movido por vez por ator
3	Um bloco pode ser pego apenas se não houver outro diretamente acima
4	Um bloco pode ser colocado numa posição apenas se houver um bloco ou mesa abaixo
5	Um ator pode se mover apenas entre posições adjacentes, uma por vez
6	Todas as entidades tem identidade única

As propriedades modeladas de domínio mais relevantes são listadas abaixo. Todas essas características são relevantes para o comportamento do sistema de planejamento e tratamento pelos processos. Essas escolhas de modelagem são para fins de simplificar a implementação, como tempo ímplicito e planejamento não-reativo, ou para fins de incluir a complexidade de domínios reais, como sistema dinâmico e observabilidade parcial.

- **Sistema Finito:** O espaço de estados do sistema é finito;
- **Sistema Observável em Posições e Proceduralmente Amostrado:** O estado completo do domínio para um determinado instante não é possível de ser observado, sendo parcialmente observável, e a amostragem é feita proceduralmente em posições do domínio;
- **Sistema Determinístico:** O resultado de ações pelos atores é absolutamente previsível;
- **Sistema Dinâmico:** Eventos exógenos podem ocorrer;
- **Objetivos Restritos:** Não existem objetivos intermediários ou restrições específicas de forma de planejamento;
- **Planos Sequenciais:** O plano de ação é uma sequência finita de ações discretas linearmente ordenadas, mesmo para o exemplo multi-agente;

- **Tempo Implícito:** Ações e eventos não possuem duração, sendo tratadas na etapa de planejamento de ação como instantâneas, não sendo parâmetro para cálculo de custo de uma ação;
- **Planejamento não-reativo:** O planejador não reaje a mudanças no domínio em tempo de planejamento.

O mundo de blocos modelado para o mundo físico e planejamento automático é composto de três classes: Posições, Blocos e o Robô. Posições e Blocos são entidades passivas, não possuindo formas de mudar o estado global do domínio, enquanto o Robô é um agente ativo e possuí métodos para promover mudança. Esses métodos de classe, atributos e relações podem ser modelados usando diagramas UML para ajudar na fase de implementação e validação de arquitetura. Os diagramas aplicados são Caso de Uso, Classes e Estado, seguindo o método de design proposto em Vaquero (2007). A representação gráfica de *snapshots* também é útil para modelagem e depuração.

A Figura 26 apresenta o diagrama de Casos de Uso do sistema, onde o Robô pode realizar oito ações distintas, a saber, pode se mover em qualquer direção, ele pode pegar blocos através de controle de garra e pode usar seu leitor RFID para identificar ou atualizar dados. As ações de movimentação e pegada são úteis no planejamento de movimento. As duas últimas ações, de leitura e escrita, são usadas quando o robô deve inspecionar ou atualizar dados PRD.

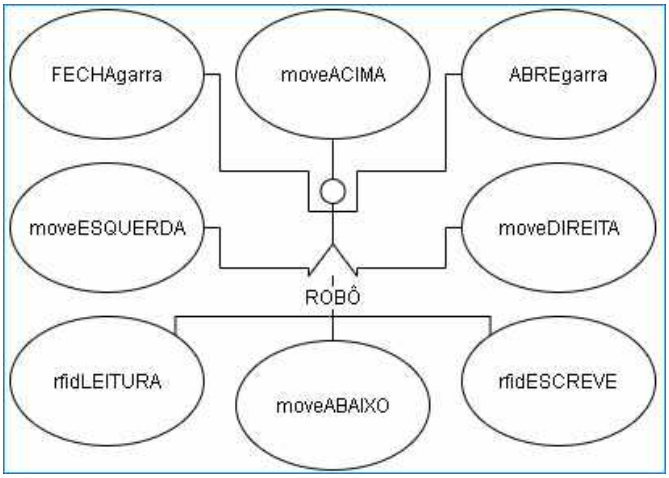

Figura 26 – Diagrama de Casos de Uso

A Figura 27 é o diagrama de Classes, uma abstração para as entidades, suas propriedades, métodos e relacionamentos no que tange o planejamento discreto de movimento. A identidade individual de cada entidade não é tratada como atributo no escopo do diagrama de Classes, pois não é diretamente relacionada com propriedades e métodos de movimento. Posição guarda os atributos para relacionamentos de posicionamento e adjacência, o Robô tem todos os métodos e um atributo para controle da garra. Blocos

não são dotados de atributos ou métodos. Na implementação da definição do domínio as classes expressas podem sofrer leves mudanças com base nas características e objetivos específicos do domínio.

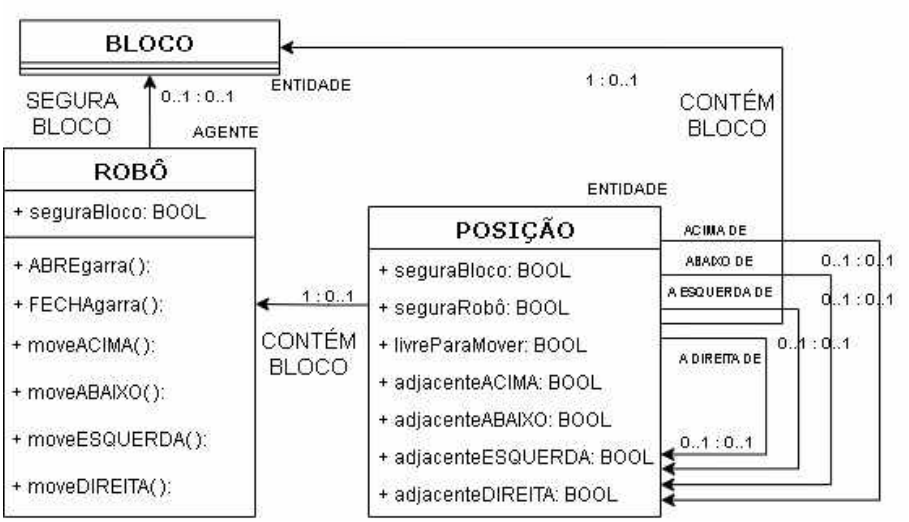

Figura 27 – Diagrama de Classes

A Figura 28 mostra o diagrama de Estado das classes Robô e Posição. Pode ser inferido que cada ação requer precondições de relacionamento e status de atributos, e pós-condições afetam apenas atributos. Os atributos são binários, logo o rastreamento de mudanças individuais é facilitado.

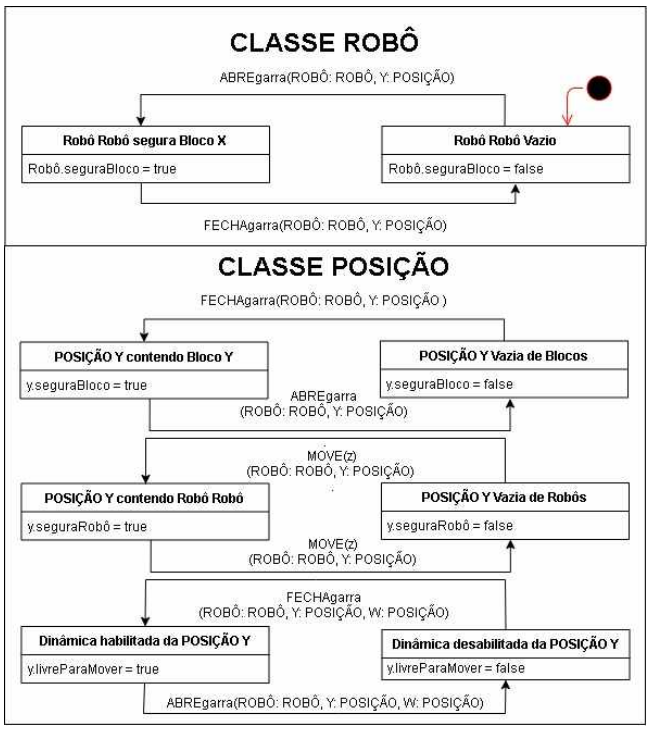

Figura 28 – Diagrama de Estado

Existem efeitos acoplados entre atributos e ações permitidas, significando que atributos de adjacência podem ser necessários, ou posições adjacentes podem ter seus atributos alterados, como efeito colateral. Um exemplo é que uma Posição pode conter o Bloco X apenas se nenhuma outra Posição ou agente estiver com posse de X, e que a Posição inferior necessita de propriedades especiais (caso de mesa) ou de segurar outro bloco. Na forma de axioma, a posição inferior ganha verdadeiro para o atributo *livreParaMover*.

O passo final proposto na representação de um domínio de planejamento é alimentar o *snapshot* inicial e objetivos. Um *snapshot* é o conhecimento de domínio sobre objetos dado um instante no tempo/eventos.

O comportamento de planejamento é baseado no modelo, logo restrições clássicas e precondições são introduzidas nas especificações de Caso de Uso. Como exemplo, um objeto da classe Robô pode pegar apenas um objeto da classe Bloco se seus atributos são garra aberta e vazia. Outra pré-condição é que um Bloco pode apenas ser colocado sobre outro Bloco ou sobre a mesa. Algumas restrições indiretamente são levantadas, já que dois Blocos não podem ocupar a mesma Posição, ou para mover entre Posições, numa ação, elas requerem adjacência direta.

Um exemplo de *snapshot* é registrado na Figura 29. Nesse *snapshot* a relação entre objetos e suas posições inicias é explícita. Definição de adjacência entre Posições é fundamental para garantir um planejamento de movimentação apropriado, e a alocação inicial dos Blocos e Robô é essencial para a geração do plano e corretude dinâmica.

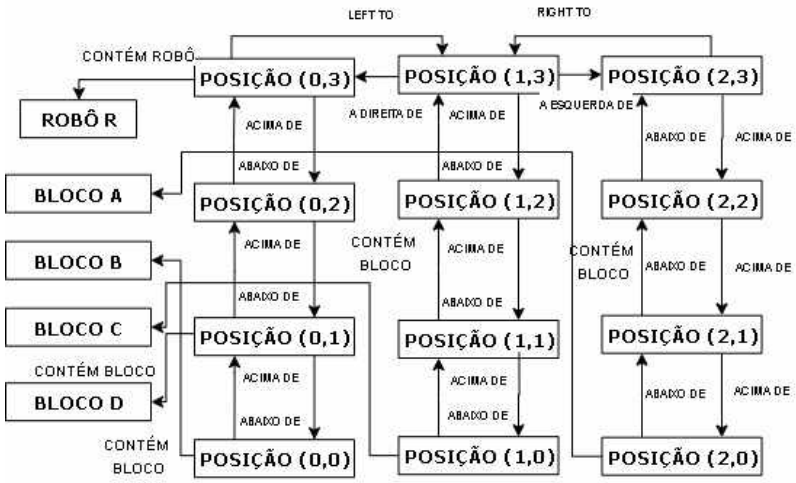

Figura 29 – *Snapshots* para visualização de estados do domínio

O *snapshot* deve ser formalizado, sendo convertido em PDDL para ser útil no planejamento. Essa conversão deve ser feita pelo microcontrolador, que utiliza de um conjunto de regras para gerar o arquivo PDDL.

4.2.2 Definição de Domínio e Problema

É importante notar que a definição de domínio final em PDDL é altamente acoplada com a arquitetura de processos, PRD e a implementação física, logo detalhes de predicados e métodos podem variar. O domínio aqui explorado foi adaptado para suporte de um único ator. A versão multiagentes agrega predicado, precondições e efeitos para evitar colisões.

```
; Mundo de Blocos Um Ator
(
  define (domain BlockWorld-Mecatronics)
  (:requirements :typing :negative-preconditions :disjunctive-preconditions
    :conditional-effects)
  (:types robo - agente
    bloco posicao - objeto)

   (:predicates
     (segurarobo ?p - posicao ?a - robo)
     (segurabloco ?x - (either braco posicao) ?b - bloco)
     (livreparapegar ?a - robo)
     (livreparamover ?p - posicao)
     (livreparamoverbloco ?p - posicao)
     (adjacenteacima ?p1 ?p2 - posicao)
     (adjacenteabaixo ?p1 ?p2 - posicao)
     (adjacenteesquerda ?p1 ?p2 - posicao)
     (adjacentedireita?p1 ?p2 - posicao)
     (empilhadobloco ?b1 ?b2 - bloco)
   )
   (:action pegabloco
     :parameters (?a - robo ?b ?bd - bloco ?p1 ?p2 - posicao)
     :precondition (and (segurabloco ?p1 ?b) (livreparamover ?p1)
       (livreparapegar ?a) (segurarobo ?p1 ?a) (adjacenteabaixo ?p1 ?p2)
       (not (livreparamoverbloco ?p2)))
     :effect (and (not (livreparapegar ?a)) (not (segurabloco ?p1 ?b))
       (segurabloco ?a ?b) (livreparamover ?p2) (livreparamoverbloco ?p1)
       (when (segurabloco ?p2 ?bd) (not (empilhadobloco ?b ?bd))))
   )
   (:action soltabloco
     :parameters (?a - robo ?b ?bd - bloco ?p1 ?p2 - posicao)
     :precondition (and (segurabloco ?a ?b) (not (livreparapegar ?a))
       (segurarobo ?p1 ?a) (adjacenteabaixo ?p1 ?p2)
       (not (livreparamoverbloco ?p2)))
     :effect (and (not (livreparamover ?p2)) (livreparapegar ?a)
     (not (segurabloco ?a ?b))
       (segurabloco ?p1 ?b) (not (livreparamoverbloco ?p1))
       (when (segurabloco ?p2 ?bd) (empilhadobloco ?b ?bd)))
   )
   (:action moveacima
     :parameters (?a - robo ?p1 ?p2 - posicao ?b - bloco)
     :precondition (and (adjacenteacima ?p1 ?p2) (segurarobo ?p1 ?a)
       (livreparamover ?p2) (or (livreparapegarbloco ?p2) (livreparapegar ?a)))
     :effect (and (not (segurarobo ?p1 ?a)) (segurarobo ?p2 ?a))
   )
       ...
)
```

Figura 30 – Especificação de Domínio com um ator

Para o domínio, precondições e efeitos condicionais são declarados. A especificação do domínio para um ator é definida na Figura 30. Dez predicados são declarados, quatro para criação de relações adjacentes entre Posições, dois para possessão de Bloco / Robô, três para restrições implícitas de movimentação e pegada de Bloco e um para indicar se

um Bloco está sobre outro Bloco. O último predicado descrito é útil pois ele permite o posicionamento relativo de objetos e facilita a implementação de estratégias de agrupamento de objetivos individuais.

Seis ações são declaradas, quatro para movimento e duas para pegar e soltar Blocos. Todas as ações tem custo unitário. As outras ações de movimentação são apenas concatenamento das quatro básicas no PSS.

O próximo passo é a definição do problema, isto é, um repositório de objetos, o *snapshot* atual e objetivos num único arquivo de problema. O repositório indica todos os objetos envolvidos. Os objetivos são predicados que indicam atributos para serem alcançados dado o conjunto de objetos.

Um arquivo de especificação de problema para um único ator é apresentado na Figura 31. Basicamente ele é dividido num cabeçalho, onde o problema recebe identidade e um domínio associado, o repositório, as condições correntes e o conjunto de objetivos.

```
; Mundo de Blocos Um Ator
(
  define (problem one-actor)
  (:domain BlockWorld-Mecatronics)
  (:objects
     blocoA blocoB blocoC blocoD - bloco
     p_0_0 p_0_1 p_0_2 p_0_3 p_0_4
     p_1_0 p_1_1 p_1_2 p_1_3 p_1_4
     p_2_0 p_2_1 p_2_2 p_2_3 p_2_4
     p_3_0 p_3_1 p_3_2 p_3_3 p_3_4 - posicao
     robo0 - robo
  )
  (:init
     ; adjacencia
     (adjacenteacima p_0_0 p_0_1)
     (adjacenteabaixo p_0_1 p_0_0)
      ...
     (adjacenteesquerda p_3_4 p_2_4)
     ; pblocos e robo
     (segurabloco p_1_1 blocoA)
     (segurabloco p_2_1 blocoB)
     (segurabloco p_3_1 blocoC)
     (segurabloco p_3_2 blocoD)
     (segurarobo p_1_4 robo0)
     (livreparapegar robo0)
     ; posicoes irrestritas
     (livreparamover p_0_1)
     (livreparamover p_0_2)
      ...
     (livreparamoverbloco p_0_1)
     (livreparamoverbloco p_0_2)
      ...
     ; pilhas
     (empilhadobloco blocoD blocoC)
  )
  (:goal (and
     (segurabloco p_2_3 blocoA)
     (segurabloco p_2_1 blocoB)
     (empilhadobloco blocoD blocoB)
     (segurarobo p_1_4 robo0))
  )
)
```

Figura 31 – Especificação de Problema com um ator

4.2.3 Modelagem Geométrica

A Figura 32 mostra o robô manipulador usado nos exemplos. É um simples robô redundante no espaço de estados de posição, com 6 graus de liberdade, dotado de articulações rotativas, atuadas por servomotores. A implementação no robô ainda não foi realizada, apenas a modelagem DGM e de exemplo de aplicação.

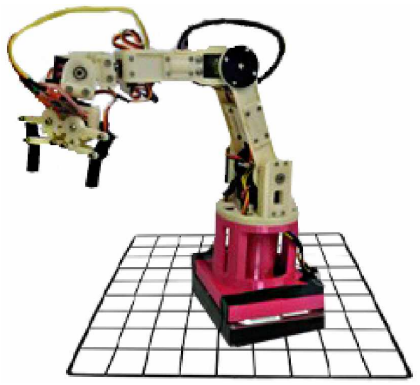

Figura 32 – Robô Manipulador didático

A Figura 33 retrata o modelo de representação do robô manipulador, onde as setas em azul retratam rotações em torno do eixo Y e as em vermelho retratam rotações em torno do eixo Z. Para fins de facilitar a formulação, a origem do sistema é fixada na base do robô, coincidente com a referencial 0, e todos os referenciais usam sistema de coordenadas semelhante, conforme Figura 34, que é usada para gerar as Equações 4.1 a 4.7.

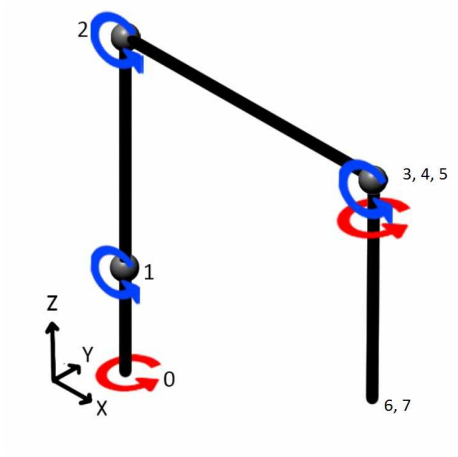

Figura 33 – Modelo de Robô Manipulador

O DGM do robô é gerado usando seu modelo de referencia estrutural. As Equações de 4.1 a 4.7 representam as transformadas para cada atuador no robô, e a transformada 4.8 se refere ao mapeamento do elemento terminal em relação à referência estática. As transformadas nas Equações 4.4, 4.5 e 4.6 referenciam atuadores distintos. A Equação 4.7 referencia o elemento terminal (garra).

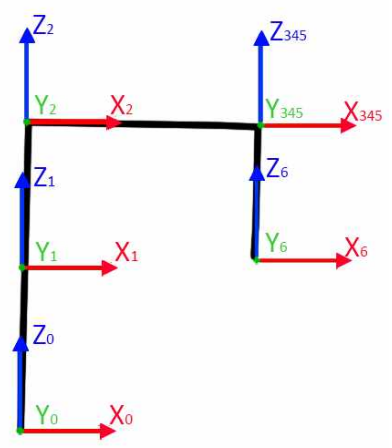

Figura 34 – Eixos de referência para cada articulação

$$ {}^{0}T_{1} = \begin{bmatrix} cos(q_1) & -sin(q_1) & 0 & 0 \\ sin(q_1) & cos(q_1) & 0 & 0 \\ 0 & 0 & 1 & 46.6 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.1} $$

$$ {}^{1}T_{2} = \begin{bmatrix} cos(q_2) & 0 & sin(q_2) & 0 \\ 0 & 1 & 0 & 0 \\ -sin(q_2) & 0 & cos(q_2) & 105.8 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.2} $$

$$ {}^{2}T_{3} = \begin{bmatrix} cos(q_3) & 0 & sin(q_3) & 0 \\ 0 & 1 & 0 & 0 \\ -sin(q_3) & 0 & cos(q_3) & 129.8 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.3} $$

$$ {}^{3}T_{4} = \begin{bmatrix} cos(q_4) & -sin(q_4) & 0 & 0 \\ sin(q_4) & cos(q_4) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.4} $$

$$ {}^{4}T_{5} = \begin{bmatrix} cos(q_5) & 0 & sin(q_5) & 0 \\ 0 & 1 & 0 & 0 \\ -sin(q_5) & 0 & cos(q_5) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.5} $$

$$ {}^{5}T_{6} = \begin{bmatrix} cos(q_6) & -sin(q_6) & 0 & 0 \\ sin(q_6) & cos(q_6) & 0 & 0 \\ 0 & 0 & 1 & 101.05 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.6} $$

$$^{6}T_{7} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.7}$$

$$^{0}T_{7} = ^{0}T_{1} \cdot ^{1}T_{2} \cdot ^{2}T_{3} \cdot ^{3}T_{4} \cdot ^{4}T_{5} \cdot ^{5}T_{6} \cdot ^{6}T_{7} \tag{4.8}$$

Onde a posição e orientação do elemento terminal podem ser calculadas conforme a Equação 4.9.

$$\begin{bmatrix} X \\ Y \\ Z \\ \psi \\ \theta \\ \phi \end{bmatrix} = \begin{bmatrix} ^{0}T_{7}(1,4) \\ ^{0}T_{7}(2,4) \\ ^{0}T_{7}(3,4) \\ atan2(^{0}T_{7}(3,2), -^{0}T_{7}(3,1)) \\ atan2(^{0}T_{7}(3,2)/cos(\psi), ^{0}T_{7}(3,3)) \\ atan2(^{0}T_{7}(1,3), -^{0}T_{7}(2,3)) \end{bmatrix} \tag{4.9}$$

O Jacobiano é calculado construindo a matriz de derivadas parciais. Cada variável de estado do elemento terminal é diferenciada por cada variável articular, conforme a Equação 4.10 aponta. A pseudo-inversa pode ser calculada conforme descrito na Secção 2.5.3.

$$J = \begin{bmatrix} \frac{\partial X}{\partial q_1} & \frac{\partial X}{\partial q_2} & \cdots & \frac{\partial X}{\partial q_6} \\ \frac{\partial Y}{\partial q_1} & \frac{\partial Y}{\partial q_2} & \cdots & \frac{\partial Y}{\partial q_6} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial \phi}{\partial q_1} & \frac{\partial \phi}{\partial q_2} & \cdots & \frac{\partial \phi}{\partial q_6} \end{bmatrix} \tag{4.10}$$

4.2.4 Geração de Trajetória

Para gerar referências de movimento entre dois pontos alguns pré-requisitos são fundamentais. O DGM do robô é conhecido; o valor corrente das variáveis articulares é conhecido; uma posição e orientação para o elemento terminal deve ser definida, dentro do espaço de trabalho do robô; e a regra de geração de trajetória e critério de granularidade devem ser definidos.

A forma mais simples de produzir trajetória é gerando uma transição linear simples, uma linha para a posição e uma revolução linear para a orientação. A representação

matricial dessa transição é representada na Equação 4.11.

$$\begin{bmatrix} X_{next} \\ Y_{next} \\ Z_{next} \\ \psi_{next} \\ \theta_{next} \\ \phi_{next} \end{bmatrix} = \begin{bmatrix} X_o \\ Y_o \\ Z_o \\ \psi_o \\ \theta_o \\ \phi_o \end{bmatrix} + t \cdot \begin{bmatrix} a \\ b \\ c \\ \gamma \\ \beta \\ \alpha \end{bmatrix} \tag{4.11}$$

Onde o lado esquerdo da Equação 4.11 representa o próximo estado objetivo. O lado direito contém o estado inicial e o coeficiente de transição linear, cujo valor é obtido adotando $t = 1$. O coeficiente t é iterativamente atualizado, in-loop, usando a Operação 4.12. Nessa operação, adotando o critério de granularidade constante, com N como granularidade, ($N \in \mathbb{N}^*$) e k como o passo objetivo atual, ($k \in \mathbb{N}^*$). O passo atual k incrementa de 1 até N, fazendo com que t incremente em passos constantes de 0 até 1.

$$t \leftarrow \frac{k}{N} \tag{4.12}$$

Adotando que os blocos sejam cubos com aresta de 30 $[mm]$, dispostos linearmente no plano XY a uma distância entre eixos verticais de 95 $[mm]$ do referencial estático do robô, em pilhas ou diretamente sobre a bancada é possível calcular posições e passo lógico-direcional. A Figura 35 traz a vista trimétrica do plano XY de trabalho, onde a posição dos blocos é disposta linearmente e homogeneamente a intervalos de 80 $[mm]$, no sentido do eixo x, com PSS $4 \cdot 4$.

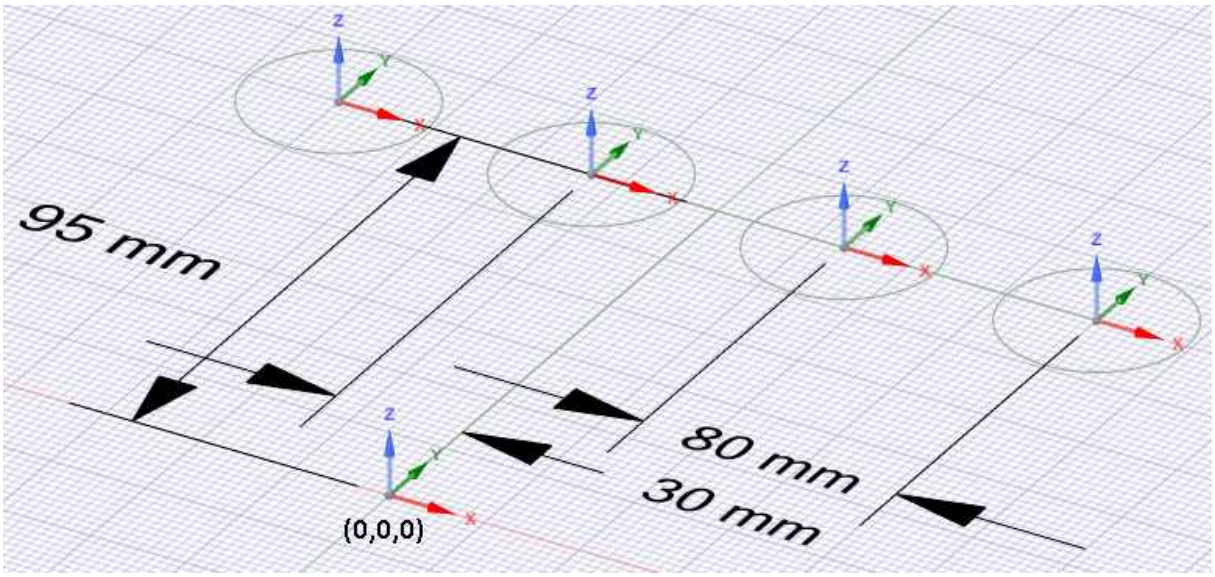

Figura 35 – Representação do plano de trabalho XY

É relevante que a atitude dos blocos respeite suas coordenadas locais para facilitar agarrar o cubo. A etiqueta RFID deve se encontrar na face XZ frontal dos cubos, de forma que facilite a leitura/escrita de dados. Também é importante que a garra venha de cima e esteja voltada para os blocos, para minimizar a chance de colisão com blocos e com a bancada, logo $[\psi, \theta] = [-90, -180]$. Determinando o primeiro ponto de referência, na coordenada (0,0) do PSS, à altura do centro geométrico do cubo e considerando os 30 $[mm]$ de aresta, obtemos o seguinte vetor cartesiano $X = [-120, 80, 15, -90, -180, 0]^T$. Com a orientação do elemento terminal definida o robô tem seu comportamento mais previsível.

Para fazer a conversão de ações em referências de posição e orientação, para uso no planejamento de trajetória é usada a Equação 4.13, fazendo o incremento sobre o estado cartesiano atual do elemento terminal.

$$\begin{bmatrix} R \\ U \end{bmatrix} = \begin{bmatrix} [80 & 0 & 0 & 0 & 0 & 0]^T \\ [\,0 & 0 & 30 & 0 & 0 & 0]^T \end{bmatrix} \tag{4.13}$$

Onde R é uma ação de movimento para a direita, U uma ação de movimento para cima e O uma ação de abertura de garra, que não passa por conversão e é interpretada diretamente pelo controlador, sendo uma simples mudança de referência para o atuador da garra. As ações reversas, mover-se para esquerda, para baixo e fechar a garra são representadas respectivamente por L, D, C e são dadas pelo decremento em valor correspondente ou interpretação contrária.

Esse sistema de posicionamento foi selecionado para facilitar as conversões cartesianas do robô, sendo arbitrário. O único cuidado fundamental é garantir que todos os pontos lógicos estão contidos no espaço de trabalho de posição e orientação.

4.2.5 Estudos de Caso

Exemplos puramente virtuais são apresentados nessa secção. Os processos de inferências e planejamento são implementados conforme descrito na Secção 4.1, localmente e usando um serviço AWS. O cálculo do vetor de referências foi feito no MATLAB© de forma desacoplada, para o primeiro exemplo.

4.2.5.1 Mono Ator

No exemplo mono ator não é utilizado o programa de percepção virtual, pois há a presença de *features* de objetivo indireto, que ainda não são suportadas pelo programa.

Para localizar todos os blocos, a PSS é exaurida. Posições são mapeadas no sentido direita e para cima, numa notação Posição(X,Y), começando em (0,0), seguindo uma máquina de estados finita. Nesse exemplo, os Blocos A, B, C, D são encontrados respectivamente nas posições (1,1), (2,1), (3,1), (3,2) conforme apresentado na Figura

36. Dentro de cada etiqueta RFID, no marcador *:current*, predicados descrevendo a situação atual podem ser encontrados, contrastando com informação na PSS. No marcador *:objective-O*, um conjunto de objetos pode ser coletado, para cada etiqueta. Nesse exemplo, cada etiqueta, tem no máximo um predicado-objetivo sobre si, para ser colocado numa determinada posição ou relativo a outro bloco, logo não há outra iteração de execução.

A Figura 36 representa o *snapshot* com a condição atual descrita na especificação de problema. Informação sobre adjacência e posições irrestritas foi escondida para reduzir espaço. As Posições inferiores, em rosa, têm atributos que fazem com que se comportem como uma mesa (tornando falsos todos os predicados de movimento), logo Blocos podem ser colocados acima e o Robô não pode se mover nessas posições. Blocos A, B, C, D são identificados em azul e o Robô em verde. Todas as relações de adjacência são implícitas pelas Posições.

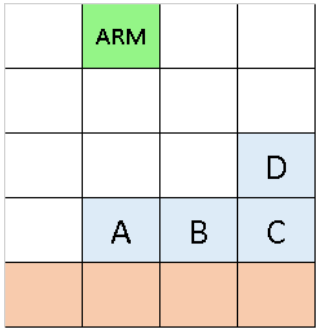

Figura 36 – *Snapshot* Inicial obtido da PSS e PRD

O objetivo, descrito em linguagem natural, é uma composição "*AND*" dos objetivos individuais, seguindo a regra GISP: Bloco D sobre Bloco B, Bloco A em (2,3), Bloco B em (2,1) e Robô em (2,4). Essa descrição de objetivo inclui posicionamento absoluto, na forma de Bloco A ocupando (2,3), posicionamento relativo, na forma de Bloco D sobre Bloco B e objetivos nulos, como o Bloco C não estabeleceu objetivo. Isso significa que configurações distintas podem satisfazer os objetivos, já que a Posição em que o Bloco C é colocado não é relevante.

Em seguida, alimentando os arquivos de Domínio e Problema, o *parser* do *Fast-Downward* converte o PDDL numa estrutura adequada a busca. Depois de gerar 2696 estados, o plano ótimo é composto de 16 passos adequados para serem introduzidos no planejador de movimento, conforme mostrado na Figura 38. A configuração ótima a ser alcançada é identificada na Figura 37.

Para calcular o plano de movimento é necessário aplicar dois filtros sobre o relatório do plano de ação gerado. O primeiro filtro usa expressões regulares para identificar ações e convertê-las, ordenadamente, em caracteres que representam a ação executada (R, L, U, D, O, C). O valor corrente é repetido como primeiro objetivo para fins de robustez e depuração, seguido da abertura da garra. O segundo filtro converte a sequência de

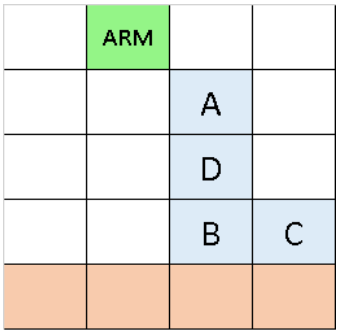

Figura 37 – *Snapshot* Final obtido da PRD

```
; Block World One-Actor
  moveabaixo robo0 p_1_4 p_1_3 blocoa
  moveabaixo robo0 p_1_3 p_1_2 blocoa
  movedireita robo0 p_1_2 p_2_2 blocoa
  movedireita robo0 p_2_2 p_3_2 blocoa
  pegabloco robo0 blocod blocoa p_3_2 p_3_1
  moveesquerda robo0 p_3_2 p_2_2 blocoa
  soltabloco robo0 blocod blocob p_2_2 p_2_1
  moveesquerda robo0 p_2_2 p_1_2 blocoa
  moveabaixo robo0 p_1_2 p_1_1 blocoa
  pegabloco robo0 blocoa blocoa p_1_1 p_1_0
  moveacima robo0 p_1_1 p_1_2 blocoa
  moveacima robo0 p_1_2 p_1_3 blocoa
  movedireita robo0 p_1_3 p_2_3 blocoa
  soltabloco robo0 blocoa blocoa p_2_3 p_2_2
  moveesquerda robo0 p_2_3 p_1_3 blocoa
  moveacima robo0 p_1_3 p_1_4 blocoa
Comprimento do plano: 16 passos.
```

Figura 38 – Plano de Ação Gerado para mono ator

caracteres, dada a posição inicial do ator, em referências de estado para o elemento terminal. O resultado final da conversão pode ser visualizado na Tabela 2, com o total de 13 ações que requerem planejamento de trajetória.

Dados os parâmetros $N = 10$, $\sigma = 1$, $E_n = 1$ o sistema teve dificuldades de convergência em alguns passos devido a dificuldade do algoritmo em lidar com oscilações, logo foi necessário calcular ponto a ponto no ambiente MATLAB© os resultados em coordenadas articulares. Partindo da condição descrita em 1-1, e o vetor de vetores de *setpoint* tem o total de 135 vetores para serem repassados ao cliente. O formato de saída do vetor de *setpoints* é representado na Tabela 3, onde o vetor de referências comanda cada atuador separadamente. É importante ressaltar que espaços proibitivos não foram modelados, logo na prática podem acontecer colisões entre a estrutura do manipulador e ela própria ou blocos adjacentes por esse algoritmo.

Tabela 2 – Sequência de referências para cálculo de trajetória

Sequência	X	Y	Z	ψ	θ	ϕ	Garra
1	-40	80	105	-90	-180	0	-
2	-	-	-	-	-	-	O
3	-40	80	75	-90	-180	0	-
4	-40	80	45	-90	-180	0	-
5	40	80	45	-90	-180	0	-
6	120	80	45	-90	-180	0	-
7	-	-	-	-	-	-	C
8	40	80	45	-90	-180	0	-
9	-	-	-	-	-	-	O
10	-40	80	45	-90	-180	0	-
11	-40	80	15	-90	-180	0	-
12	-	-	-	-	-	-	C
13	-40	80	45	-90	-180	0	-
14	-40	80	75	-90	-180	0	-
15	40	80	75	-90	-180	0	-
16	-	-	-	-	-	-	O
17	-40	80	75	-90	-180	0	-
18	-40	80	105	-90	-180	0	-

Tabela 3 – Vetores de referências articulares para controle robótico

Vetor	q_1	q_2	q_3	q_4	q_5	q_6	q_7
1-1	116.565	-14.834	78.698	359.999	-283.865	206.656	-
...	...	...	...	...	...	...	...
2	-	-	-	-	-	-	O
3-1	296.523	-75.057	80.678	179.747	185.609	-333.730	-
3-2	116.073	-283.328	277.199	355.819	-174.061	-338.085	-
3-3	115.863	78.291	-84.780	354.183	-173.784	-339.921	-
...	...	...	...	...	...	...	...
6-10	326.043	351.400	-70.368	180.388	-259.534	-303.885	-
7	-	-	-	-	-	-	C
8-1	215.343	-70.379	53.065	-359.090	196.947	-233.787	-
8-2	-142.620	-71.537	57.532	181.034	-193.687	-51.616	-
...	...	...	...	...	...	...	...
18-9	213.551	-70.246	49.885	-359.421	-159.930	-235.906	-
18-10	115.943	-286.604	-78.883	353.831	185.240	-340.202	-

Quanto ao controle do robô, usando um interpretador é possível atender aos vetores de *setpoint*, usando as referências em degrau e um intervalo fixo de mudança de vetor de referências. Articulações não referidas mantém o valor corrente. No robô do exemplo são usados servomotores com correção de ângulo implementada em *hardware*, logo não é necessário o desenvolvimento de leis de controle ou filtros no *firmware* do cliente.

4.2.5.2 Multi Ator

No exemplo com múltiplos atores não será tratado o planejamento de trajetória nem do controle, pois apesar do problema de choque mecânico ser tratado pelas regras de domínio adicionais no caso multi agente, seria necessário ajustar o posicionamento do PSS para o segundo robô, devido ao consequente deslocamento. Além disso a efetuação de ação paralela e concorrente eficiente requer comunicação e um *middleware* eficaz em sincronizar os atores.

Usando o programa de percepção de ambiente virtual, os arquivos com informação do domínio são dispostos conforme a Figura 39. Em *prebuilt* é criada uma definição da PSS em *PSS.def* e dos atores no formato PRD em *arm*.def*. No domínio são declarados 4 blocos distintos, nos arquivos **.in*. O processo *process* recebeu o argumento *2ArmsProblem* para gerar o arquivo no diretório *output*.

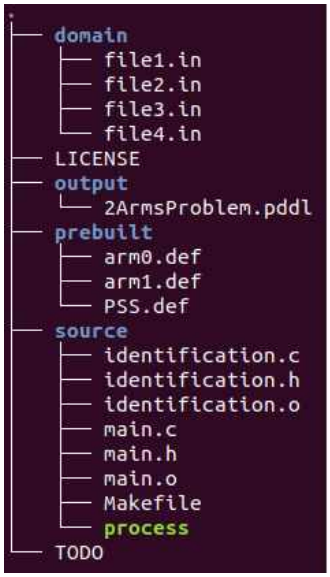

Figura 39 – Árvore de projeto populada para o exemplo

As respectivas PRDs virtuais dos blocos de agentes e atores são dadas seguindo o modelo dado na Figura 40, com marcadores e predicados que dizem respectivamente sobre a identidade, estado corrente e objetivo individual do elemento. O processo identifica todos os arquivos nos diretórios com as extensões e subnomes adequados para a geração de código PDDL. O arquivo PDDL gerado está pronto para ser alimentado numa máquina de planejamento de ação.

A Figura 41 representa o *snapshot* inicial, após coletar e inferir dados da PRD virtual. As condições objetivo são descritas como uma composição "*AND*": Bloco A em (3,1), Bloco B em (1,1), Bloco C em (2,1), Bloco D em (1,2), ARM0 em (3,4) e ARM1 em (0,4). Como todos os objetivos individuais são absolutos em natureza, o *snapshot* objetivo pode ser diretamente representado (Figura 42).

```
:basic_info
{
   ID(arm0)
}
:current
{
  position(place_1_4)
}
:objective-0
{
   position(place_3_4)
}
```

Figura 40 – Modelo de PRD do exemplo

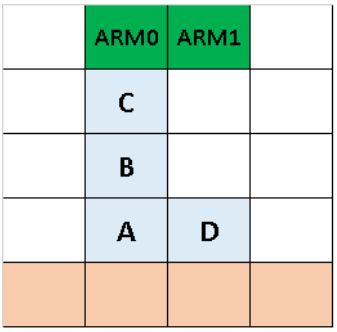

Figura 41 – *Snapshot* Inicial com dois atores

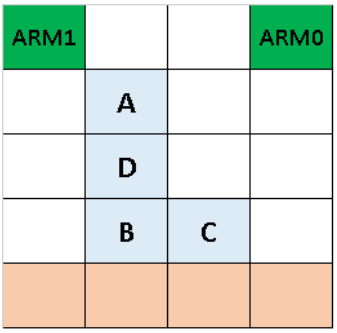

Figura 42 – *Snapshot* Final com dois atores

Após gerar 2.137.032 estados, o plano ótimo é composto de 36 passos totais, sem paralelização de ação dos atores. Aplicando o filtro de expressões regulares adaptado para multiatores obtem-se o vetor sequencial de ações apresentado na Figura 43, onde o ator correspondente à ação é apresentado em conjunto, num par ordenado, usando "−" para separar ator de ação, e "|" para indicar o próximo par ator-ação sequencial. Conforme mencionado na secção 3.5.2 esse planejamento para multi atores só é possível para agentes colaborativos, pois não há disputa de recursos.

```
arm0-d|arm0-c|arm1-d|arm1-d|arm0-r|arm1-d|arm0-d|arm1-c|
arm1-r|arm0-d|arm1-u|arm0-o|arm1-l|arm1-o|arm1-l|arm1-c|
arm0-l|arm0-c|arm0-l|arm0-u|arm0-u|arm1-d|arm0-r|arm1-o|
arm1-u|arm1-r|arm1-c|arm1-l|arm1-o|arm0-o|arm0-r|arm1-u|
arm1-l|arm0-u|arm0-r|arm1-u|
```

Figura 43 – Plano multiagentes apresentado sequencialmente

4.2.6 Discussões

Na primeira implementação o servidor foi emulado, sendo hospedado num computador com Intel(R) CoreTM i5-4210U CPU @ 1.70 GHz x86_64, caches: L1d com 32 kB; L1i com 32 kB; L2 com 256 kB; L3 com 3072 kB. 4.0 GB DDR3 RAM e memória secundária SSD $M2$ 240 GB usando Linux com kernel genérico na versão 4.15.0-118. O primeiro cenário teve tempo de busca de 0.004 segundos, enquanto no segundo foram 0.68 segundos em média.

Na segunda implementação foi usado um servidor AWS EC2©, sendo hospedado numa imagem de máquina virtual com kernel 5.3.0-1035-aws, com acesso a um único core de uso não-exclusivo e recursos de 1 GB de memória principal e 8.0 GB de memória secundária SSD gp2. O *hardware* é um Intel(R) Xeon(R) CPU E5-2676 v3 @ 2.4 GHz x86_64, com os mesmos níveis e valores de cache. O primeiro cenário teve tempo de busca de 0.004 segundos, enquanto no segundo foram 0.55 segundos em média. A latência de comunicação com o servidor, hospedado na Carolina do Norte (Estados Unidos), é da ordem de 200 ms.

A forma de declaração tem influência sobre a estrutura de busca resultante, mas apesar disso resultados ainda podem ser comparados, pelo fato da magnitude desse efeito ser relativamente baixa. Comparando os resultados do planejamento de ação no exemplo mono e multi ator, pode ser inferido que um ganho adicional de complexidade na descrição do domínio e a adição de um ator fez com que o problema se torne significativamente mais custoso de se resolver. O número de estados gerados e o tempo de busca cresceu em algumas ordens de magnitude, enquando a profundidade mais que dobrou, mesmo usando uma heurística mais adequada para a dimensão do problema.

Quanto às opções de planejamento discreto, o algoritmo de busca A* é usado acoplado a heurística h_{max} (KEYDER; GEFFNER, 2008), uma heurística genérica, para a solução do caso monoator. Outra razão para evitar o uso explícito de axiomas é que h_{max} não mantém suas propriedades sob uso de axiomas (admissível, consistente e seguro). No caso multi ator a heurística PDB obteve o desempenho superior, encontrando a solução ótima com a melhor performance.

Essas opções foram especificadas pois a busca A* garante geração de plano ótimo, sob uso de heurística admissível, e a heurística h_{max} teve a melhor performance para o exemplo monoator dentro das opções, enquanto PDB teve desempenho superior no exemplo multiator. Geração de planos ótimos é relevante pois é esperado que o processo gargalo no consumo de tempo para essa aplicação seja a movimentação física, logo o investimento em encontrar uma solução ótima é justificado.

Especificações sem coerência são, ainda, um problema aberto na identificação, em prévia a serem enviados ao planejador, e serão discutidos num trabalho futuro. No entanto, o comportamento normal do planejador é acusar erros no arquivo de problemas, caso ele não possa ser resolvido, ou caso um predicado que referencie o mesmo objeto em *:init* ou *:goal* seja declarado múltiplas vezes.

No planejamento de trajetória, no exemplo mono ator, houve um grande número de iterações de otimização. No ambiente MATLAB©, dentro da máquina descrita na primeira implementação o tempo total de execução é elevado e bastante inconsistente, devido ao uso da linguagem interpretada, de funções simbólicas e da natureza do algoritmo de otimização. Uma implementação futura mais robusta é necessária.

Os vetores de ação finais são possivelmente longos, logo cuidados para que não ultrapassem a memória *buffer* do cliente podem ser necessários em situações de controle muito extensas.

Da percepção, no exemplo multi ator, a versão atual tem pouca flexibilidade quanto aos predicados, mas já possui suporte para múltiplas entidades. As *strings* de referência para identificação e inferência estão internalizadas como *hardcode* no programa e os *buffers* de entrada são constantes. O arquivo resultante do processo de inferência se comporta de forma semelhante ao desenhado manualmente. Na primeira máquina o tempo de execução total do processo é da ordem de 0.003 segundos e na segunda de 0.002 segundos, que mostra que a aplicação de um processo semelhante num microcontrolador não é necessariamente proibitivo. O programa consome 180 kB de memória usando *buffers* generosos e predominância de alocação estática.

Quanto a identificação e tratamento de exceções, os tipos de exceções são:

1. Ausência de objetos mencionados como parte dos objetivos;
2. Marcadores e predicados fora do *thesaurus* implementado;
3. Objetivos não-factíveis devido a impossibilidade de realização física;
4. Diferença entre estado alcançado e estado atual, ou entre os dados de posição lógica e física, da PRD e PSS;

5. Eventos exógenos que alteram o estado do sistema, na forma de mudança de posicionamento, introdução ou remoção de objetos;

6. Memória corrompida, leitura ou escrita falha na identificação ou atualização;

7. Falhas de *hardware* do agente ativo ou presença de obstáculos/ruído que comprometam a capacidade de atuação.

Para todas as exceções pode ser necessário como tratamento a notificação de um supervisor. As exceções do tipo 1 podem ser relacionadas a falha de identificação, falha (prévia) no conteúdo PRD ou extravio de um objeto, o tratamento é refazer o processo de identificação. Nos tipo 2 e 3 a identificação é dependente do módulo de análise de exceções durante o processo de identificação, o tratamento é adotar um critério padrão ou suspender o processo. Exceções do tipo 4 podem ser causadas por primeiro contato com um sistema já em discordância ou como subefeito do tipo 5. O tratamento é reiterar os processos de identificação, planejamento e atuação. No tipo 5 as alterações podem não ser necessariamente observadas, já que a amostragem é feita em episódios em posições discretas. Isso significa que é possível que um agente externo atue sobre o sistema sem ser percebido, desde que o produto de suas ações não esteja ma fila corrente de amostragem.

As exceções do tipo 6 e 7 são relacionadas a falhas de *hardware* nos agentes passivos ou ativos. No tipo 6 é indicativo que o sistema RFID falhou, seja devido a problemas na memória ou na comunicação. No tipo 7 são problemas no *hardware* do agente ativo, seja por problemas de comprometimento estrutural, dinâmico ou controle. A identificação de falhas do tipo 6 sem o auxílio de um supervisor fica a cargo de lógica no *firmware*, o que torna algumas exceções não observáveis. No tipo 7 a identificação fica a cargo de sensores e lógica no *firmware*. O tratamento é a parada do sistema.

5 CONCLUSÃO E TRABALHOS FUTUROS

A arquitetura proposta integra planejadores automáticos genéricos com atores robóticos criando um CPS usando PRD e PSS. É uma ponte entre planejamento automatizado e controle adaptativo a eventos discretos onde pode ser aplicado em diversos domínios de forma inovadora para atingir objetivos multiagente, aumentando a autonomia e reduzindo a presença humana.

A PRD é uma estrutura de dados flexível para tomada de decisão de entidades passivas. Suas limitações são devido ao tamanho de memória RFID, porque os marcadores atuais tem capacidade muito limitada, problemas de leitura RFID, e a necessidade de uma estrutura PDDL especializada no leitor para o processamento correto de cada etiqueta. A arquitetura do controlador descrito para implementação na versão completa é composto por muitos tipos de processos, onde cada um deles pode ser otimizado. Essa otimização, em contrapartida, pode ser custosa para modelagem ou especializada para famílias de casos, logo obter boas métricas e boa visão para melhoria do sistema é desafiador.

A arquitetura proposta ainda não é genérica e robusta o suficiente para ser utilizada numa gama de domínios, especificamente aqueles que lidam com múltiplos atores decentralizados ou abstrações de ações que não podem ser diretamente refinadas para atuação, requisitando diversos níveis de busca. O planejador também foi construindo buscando simplicidade na dinâmica discreta, planejamento de trajetória e execução, logo uma generalização robusta para sistemas híbridos é um passo relevante para suportar domínios com dinâmicas contínuas que não podem facilmente ser modelados em termos de transições discretas. Uma interface de supervisão não foi modelada, logo a saída de dados interpretáveis por pessoas e a capacidade de intervenção de um técnico são funções úteis para a arquitetura.

Um processador de marcadores absolutamente genérico não é possível, não porque ele necessitaria de muita padronização, mas sim porque os processos em si variam muito, logo uma abstração num determinado contexto pode ser diferente de outro contexto. O que é mais palpável é a criação de processadores e máquinas de inferência para famílias de aplicações, onde bases de dados e regras são mais facilmente compartilháveis devido às características de domínio e *thesaurus* comum.

Essa abordagem lida com observação parcial devido às características de leitura RFID. Isso causa problemas de incerteza, onde o domínio pode ter sofrido com distúrbios por forças externas entre ações. O sistema lida com observabilidade parcial levantando exceções sempre que algo está em desacordo ou é não-factível, retornando a um ponto prévio de controle para corrigir. Isso pode ser melhorado adicionando sensores e mecanismos

de inferência e percepção mais poderosos para melhor rastrear objetos e detectar exceções mais cedo.

Quanto ao planejamento discreto, comparando as três implementações de acesso ao serviço, local cabeada, por página web e por processo vigilante, cada uma apresenta uma situação de uso distinta. A conexão cabeada restringe o uso para servidores localizados num espaço físico próximo da operação da máquina, mas evita a sensibilidade às incertezas de uma rede externa. A implementação por página WEB facilita a supervisão, pois toda a informação relevante de entrada, planejamento e saída pode ser exposta na página, e a implementação por processo vigilante é mais adaptável e escala mais facilmente para uso com múltiplos clientes.

Vários desafios surgem na implementação de cada processo da arquitetura PRD. Naturalmente, adaptações e especializações ocorrem para melhor se encaixarem noutros processos ou para simplificar a implementação, conforme visto no módulo de planejamento e driver de controle. E a definição do sistema em si deve ser revisada ou complementada para garantir comportamento correto. As especificações para implementação dos blocos de planejamento de trajetória e controle são específicas do *hardware* e espaço da aplicação.

A principal contribuição é a integração de RFID e planejador automático, por meio da PRD, mostrando com clareza que o planejamento automático pode ser aplicado na prática, e em casos onde soluções regulares, normalmente máquinas de estado, não são adequadas.

5.1 TRABALHOS FUTUROS

Os trabalhos futuros se separam em melhoria de implementações, complementação da arquitetura, realização de novos módulos e estudo de novos métodos para ganho de flexibilidade ou performance.

- Adaptação da arquitetura para lidar mais facilmente com sistemas de dinâmica complexa e sistemas de dinâmica híbrida;
- Desenvolvimento de uma interface de supervisão;
- Sobre a inferência na percepção, quanto a geração de código PDDL, a criação de analisadores léxicos e sintáticos para generalização da geração dos problemas PDDL podem ser usados os programas *flex(1)* e *bison(1)* do conjunto de ferramentas GNU (ANTHONY, 2014). Para tal é necessário estabelecer e padronizar a linguagem de marcação de dados da PRD e a representação intermediária GISP. Além disso é importante a inclusão de um processo para detecção de exceções, via análise semântica e falhas lógicas. O programa gerado pode ser adaptado para funcionamento cíclico num microcontrolador, desde que os custos de tempo e espaço não sejam elevados;

- Sobre o planejamento enquanto serviço remoto, ainda é necessário uma série de cuidados e implementações robustas para tornar o serviço aberto, como da implementação de um servidor *daemonizado* com suporte a múltiplos clientes e suporte a mais parâmetros de entrada. Uma alternativa a manutenção ativa de uma infraestrutura é o uso do paradigma *serverless*, implementado pelo AWS Lambda©, onde o serviço, com suas dependências, é chamado como uma função transparente por um processo local, não sendo necessário se preocupar onde e como o serviço foi processado;

- Da interface entre planejamento de ação e planejamento de trajetória deve-se implementar um processo "cola fina", usado para converter o plano de saída num vetor de refereñcias de trajetória. Essa metodologia é importante para evitar o acoplamento de implementação entre os módulos, o que reduziria o potencial de reuso independente e dificulta a manutenção. Na prática é mais simples criar interfaces pequenas para conectar programas, como é o caso das APIs *socket* e USB;

- Quanto ao planejador de movimento, existem problemas na qualidade do algoritmo de otimização, que não lida bem com oscilações, e de características práticas do domínio físico, como da modelagem de configurações proibitivas e colisão entre objetos. Há também de se ver quanto a identificação de coordenadas fora do espaço de estados e sobre políticas de flexibilização da solução. É proposto o estudo de pacotes do *middleware* ROS (KOUBAA et al., 2019), do inglês *Robot Operating System*, para identificar algoritmos, formas de tratamento e implementações em linguagens mais eficientes, como o C++, para ser usado em tempo real;

- Vaquero (2010) direciona para a análise pós design de aplicações de IA, e o ciclo de refinamento, quer dizer, uma direção para melhorar a definição de domínios. Pós análise pode ser útil na melhoria de todos os processos descritos;

- Adição de sensores e sistemas complementares, especialmente no que tange a percepção e identificação de exceções;

- A implementação física completa e especificação/implementação detalhada de todos os módulos e ferramentas.

REFERÊNCIAS

AHO, A. V. et al. **Compilers: Principles, Techniques, Tools**. 2nd. ed. [S.l.]: Academic Press, 2006. Citado na página 47.

AI, Y.; PENG, M.; ZHANG, K. Edge computing technologies for internet of things: a primer. **Digital Communications and Networks**, v. 4, n. 2, p. 77 – 86, 2018. ISSN 2352-8648. Disponível em: <http://www.sciencedirect.com/science/article/pii/S2352864817301335>. Citado na página 22.

Aijaz, A.; Sooriyabandara, M. The tactile internet for industries: A review. **Proceedings of the IEEE**, v. 107, n. 2, p. 414–435, 2019. Citado na página 22.

ANGELES, J. **Fundamentals of Robotic Mechanical Systems: Theory, Methods, and Algorithms, Second Edition**. Montreal, Canada: Springer, 2002. Citado nas páginas 30, 31 e 32.

ANTHONY, A. A. **Compiler Construction using Flex and Bison**. [S.l.]: Walla Walla College, 2014. Citado na página 76.

ASHTON, K. et al. That 'internet of things' thing. **RFID journal**, v. 22, n. 7, p. 97–114, 2009. Citado na página 20.

BARANAUSKAS, M. C. C. Procedimento, função, objeto ou lógica? linguagens de programação vistas pelos seus paradigmas. **Computadores e Conhecimento: Repensando a Educação. Campinas, SP, Gráfica Central da Unicamp**, 1993. Citado na página 25.

BARRETO, A. R. **SISTEMA CIBER-FÍSICO BASEADO NA INTEGRAÇÃO PNRD E IPNRD UTILIZANDO CLOUD COMPUTING**. Dissertação (monography) — Univerdade Federal de Uberlândia, Faculdade de Engenharia Mecânica, 2019. Citado na página 23.

BHARDWAJ, S. et al. Cloud computing: A study of infrastructure as a service (iaas). **International Journal of Engineering and Information Technology (IJEIT)**, Vol. 2, Issue 1, p. pp.60–63, 2010. Citado na página 21.

BOCHMANN, G. V. Protocol specification for osi. **Computer Networks and ISDN Systems**, v. 18, n. 3, p. 167 – 184, 1990. ISSN 0169-7552. Application of Formal Techniques ot the OSI Protocols. Disponível em: <http://www.sciencedirect.com/science/article/pii/016975529090132C>. Citado na página 45.

BUSS, S. Introduction to inverse kinematics with jacobian transpose, pseudoinverse and damped least squares methods. **IEEE Transactions in Robotics and Automation**, v. 17, 05 2009. Citado na página 33.

CONNER, D. E. **Interligação de Redes com TCP/IP: Princípios, Protocolos e Arquitetura**. 3. ed. [S.l.]: Elsevier, 2015. ISBN 978-85-352-7863-7. Citado na página 54.

COULORIS, G.; DOLLIMORE, J.; KINDBERG, T. **Sistemas Distribuídos: Conceitos e Projeto**. 4th. ed. [S.l.]: Bookman Companhia, 2007. ISBN 8560031499, 9788560031498. Citado nas páginas 21 e 45.

DEVELOPERS, F.-D. G. **Fast-Downward Official Documentation**. [S.l.], 2018. Citado na página 29.

DODSON, B. et al. Secure, consumer-friendly web authentication and payments with a phone. In: GRIS, M.; YANG, G. (Ed.). **Mobile Computing, Applications, and Services**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012. p. 17–38. ISBN 978-3-642-29336-8. Citado na página 18.

ELLUL, K. et al. Regular expressions: New results and open problems. **J. Autom. Lang. Comb.**, Otto-von-Guericke-Universitat, DEU, v. 9, n. 2–3, p. 233–256, set. 2004. ISSN 1430-189X. Citado na página 48.

EMDEN, M. H. V.; KOWALSKI, R. A. The semantics of predicate logic as a programming language. **J. ACM**, Association for Computing Machinery, New York, NY, USA, v. 23, n. 4, p. 733–742, oct 1976. ISSN 0004-5411. Disponível em: <https://doi.org/10.1145/321978.321991>. Citado na página 24.

FADALI, M. S.; VISIOLI, A. **Digital Control Engineering: Analysis and Design**. [S.l.]: Academic Press, 2013. ISBN 0123943914, 9780123943910. Citado na página 51.

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. Citado na página 25.

FERREIRA, W. R. B. **Robotics Trajectory Planning Using B-spline**. Dissertação (Mestrado) — Universidade Federal de Uberlândia, Uberlândia, 2011. Citado na página 34.

FIKES, R. E.; NILSSON, N. J. Strips: A new approach to the application of theorem proving to problem solving. **Artificial Intelligence**, v. 2, n. 3, p. 189 – 208, 1971. ISSN 0004-3702. Disponível em: <http://www.sciencedirect.com/science/article/pii/0004370271900105>. Citado na página 25.

FINKENZELLER, K. **RFID handbook: fundamentals and applications in contactless smart cards, radio frequency identification and near-field communication**. [S.l.]: John Wiley & Sons, 2010. Citado nas páginas 18 e 19.

FONSECA, J. P. da S. et al. Planpas: Plc and automated planning integration. **International Journal of Computer Integrated Manufacturing**, Taylor Francis, v. 29, n. 11, p. 1200–1217, 2016. Citado na página 14.

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. Citado na página 24.

Galloway, B.; Hancke, G. P. Introduction to industrial control networks. **IEEE Communications Surveys Tutorials**, v. 15, n. 2, p. 860–880, 2013. Citado na página 22.

GHALLAB, M.; NAU, D.; TRAVERSO, P. **Automated Planning: Theory and Practice**. [S.l.]: Morgan Kauffman, 2004. Citado na página 14.

GHALLAB, M.; NAU, D.; TRAVERSO, P. The actor's view of automated planning and acting: A position paper. **Artificial Intelligence**, Elsevier, v. 208, p. 1–17, 2014. ISSN 0004-3702. Citado na página 26.

GHALLAB, M.; NAU, D.; TRAVERSO, P. **Automated Planning and Acting**. [S.l.]: Cambridge University Press, 2016. Citado nas páginas 7 e 46.

GIL, Y.; RATNAKAR, V. A comparison of (semantic) markup languages. In: . [S.l.: s.n.], 2002. p. 413–418. Citado na página 37.

GILCHRIST, A. **Industry 4.0: the industrial internet of things**. [S.l.]: Apress, 2016. Citado na página 20.

GIMENEZ, O.; JONSSON, A. The complexity of planning problems with simple causal graphs. **Journal of Artificial Intelligence Research**, v. 31, p. 319 – 351, 2008. Citado na página 27.

GUERIN, J. et al. Unsupervised robotic sorting: Towards autonomous decision making robots. **International Journal of Artificial Intelligence and Applications**, 2018. Citado na página 13.

GUPTA, N.; NAU, D. S. On the complexity of blocks-world planning. **Artificial Intelligence**, v. 56, n. 2, p. 223–254, 1992. ISSN 0004-3702. Citado na página 55.

HASLUM, P. et al. **An Introduction to the Planning Domain Definition Language**. 1. ed. [S.l.]: Morgan Claypool Publishers, 2019. ISBN 0-321-19368-7. Citado na página 26.

HOFER, L. **Decision-making algorithms for autonomous robots**. Tese (Doutorado) — Université de Bordeaux, 2017. Citado na página 13.

Jazdi, N. Cyber physical systems in the context of industry 4.0. **2014 IEEE International Conference on Automation, Quality and Testing, Robotics**, p. 1–4, 2014. Citado na página 13.

JIN, M. et al. Jacobian-Based Topology Optimization Method Using an Improved Stiffness Evaluation. **Journal of Mechanical Design**, v. 140, n. 1, 11 2017. ISSN 1050-0472. 011402. Disponível em: <https://doi.org/10.1115/1.4038332>. Citado na página 34.

KELLY, R.; DAVILA, V.; LORIA, A. **Control of Robot Manipulators in Joint Space**. [S.l.]: Springer, 2005. Citado na página 51.

KEYDER, E.; GEFFNER, H. Heuristics for planning with action costs revisited. **ECAI 2008 - 18th European Conference on Artificial Intelligence, Patras, Greece**, 2008. Citado na página 72.

KORF, R. E.; REID, M. On the complexity of admissible heuristic search. **AAAI-98 Proceedings**, 1998. Citado na página 39.

KOUBAA, A. et al. **Robot Operating System - The Complete Reference**. 4. ed. [S.l.]: Springer, 2019. ISBN 978-3-030-20189-0. Citado na página 77.

KULKARNI, G. et al. Cloud computing-infrastructure as a service amazon ec2. **International Journal of Engineering Research and Applications (IJERA)**, Vol. 2, Issue 1, p. pp.117–125, 12 2011. Citado na página 22.

KUMAR, G. S. et al. Path planning algorithms: A comparative study. In: . [S.l.: s.n.], 2011. Citado na página 28.

LANGLEY, D. J. et al. The internet of everything: Smart things and their impact on business models. **Journal of Business Research**, 2020. ISSN 0148-2963. Disponível em: <http://www.sciencedirect.com/science/article/pii/S014829631930801X>. Citado na página 21.

LOTLIKAR, T. et al. Comparative study of barcode , qr-code and rfid system. **Int.J.Computer Technology Applications (IJCTA)**, v. 4, p. 817–821, 2013. ISSN 2229-6093. Citado na página 19.

LYNCH, K. M.; PARK, F. C. **MODERN ROBOTICS MECHANICS, PLANNING, AND CONTROL**. [S.l.]: Cambridge Press, 2017. Citado nas páginas 30, 32 e 51.

MARINHO, M. et al. Dynamic active constraints for surgical robots using vector field inequalities. **IEEE TRANSACTIONS ON ROBOTICS**, v. 35, n. 5, October 2019. Citado na página 34.

MARTINEZ, J. S.; SILVA, J. R. A new hierarchical approach to requirement analysis of problems in automated planning. **Engineering Applications of Artificial Intelligence**, v. 81, p. 373 – 386, 2019. ISSN 0952-1976. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0952197619300430>. Citado na página 26.

MCDERMOTT, D. et al. **PDDL: The Planning Domain Definition Language**. [S.l.], 1998. Citado na página 25.

MELL, P.; GRANCE, T. et al. The nist definition of cloud computing. Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology, 2011. Citado na página 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. Citado na página 25.

MICHNIEWICZ, J.; REINHART, G. Cyber-physical-robotics – modelling of modular robot cells for automated planning and execution of assembly tasks. **Mechatronics**, v. 34, p. 170 – 180, 2016. ISSN 0957-4158. System-Integrated Intelligence: New Challenges for Product and Production Engineering. Citado na página 13.

Murata, T. Petri nets: Properties, analysis and applications. **Proceedings of the IEEE**, v. 77, n. 4, p. 541–580, 1989. Citado na página 23.

MUROFUSHI, R. H. **Desenvolvimento de um sistema de posicionamento interno baseado na potência do sinal de resposta de um sistema RFID**. Dissertação (Mestrado) — Univerdade Federal de Uberlândia, Faculdade de Engenharia Mecânica, 2016. Citado na página 19.

NASCIMENTO, L. B. et al. Planejamento de caminho para sistemas robóti-cos autônomos. Sociedade Brasileira de Computação (SBC), 2020. Citado na página 34.

NAU, D. S. et al. Blended planning and acting: Preliminary approach, research challenges. In: BONET, B.; KOENIG, S. (Ed.). **INPROCEEDINGS of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA**. AAAI Press, 2015. p. 4047–4051. Disponível em: <http://www.aaai.org/ocs/index.php/AAAI/AAAI15/paper/view/9746>. Citado nas páginas 7, 14 e 15.

NUNES, J. d. S. **Uma introdução concisa sobre os fundamentos da Lógica**. [S.l.: s.n.], 2020. 367 p. Citado na página 24.

ORGANICK, E. I. A fortran iv primer. **Addison-Wesley**, p. p. 42, 1966. Citado na página 31.

Pan, J.; McElhannon, J. Future edge cloud and edge computing for internet of things applications. **IEEE Internet of Things Journal**, v. 5, n. 1, p. 439–449, 2018. Citado na página 21.

PATRIKSSON, M. et al. **An Introduction to Continuous Optimization - Foundations and Fundamental Algorithms**. [S.l.: s.n.], 2016. ISBN 978-91-44-11529-0. Citado na página 32.

PRIEDITIS, A. E. Machine discovery of effective admissible heuristics. **Machine Learning**, Kluwer Academic Publishers, Boston, v. 12, p. 117 – 141, 2008. Citado na página 28.

ROJKO, A. Industry 4.0 concept: Background and overview. **International Journal of Interactive Mobile Technologies**, 2017. Citado na página 13.

RUSSELL, S.; NORVIG, P. **Artificial Intelligence: A Modern Approach**. 3rd. ed. Upper Saddle River, NJ, USA: Prentice Hall Press, 2010. ISBN 0136042597, 9780136042594. Citado nas páginas 14, 26 e 29.

SABINO, V. C. **Um estudo sistemático de licenças de software livre**. Dissertação (Mestrado), 2011. Citado na página 54.

SAGLIANO, M.; THEIL, S. Hybrid jacobian computation for fast optimal trajectories generation. 08 2013. Citado na página 34.

SANDLER, B.-Z. **ROBOTICS - Designing the Mechanisms for Automated Machinery**. San Diego, California: ACADEMIC PRESS, 1999. Citado na página 32.

SCHLUTER, M. et al. Vision-based identification service for remanufacturing sorting. **Procedia Manufacturing**, v. 21, p. 384 – 391, 2018. ISSN 2351-9789. 15th Global Conference on Sustainable Manufacturing. Disponível em: <http://www.sciencedirect.com/science/article/pii/S2351978918301744>. Citado na página 19.

SEIPP, J.; ROGER, G. Fast downward stone soup 2018. In: . [S.l.]: Proc. Fourth International Planning Competition: International Conference on Automated Planning and Scheduling, 2018. Citado na página 54.

Silva Miranda, D. S.; de Souza, L. E.; Sousa Bastos, G. A rosplan-based multi-robot navigation system. In: **2018 Latin American Robotic Symposium, 2018 Brazilian Symposium on Robotics (SBR) and 2018 Workshop on Robotics in Education (WRE)**. [S.l.: s.n.], 2018. p. 248–253. Citado na página 26.

SILVA, T. et al. Internet of things: Concepts, applications, challenges and trends. In: . [S.l.: s.n.], 2016. p. 129–149. Citado na página 20.

SOUZA, G. A.; BRITO, H.; TAVARES, J. Automatic column assignment for block world domain with *PNRD/iPNRD*. In: SBA. **14th Brazilian Symposium on Intelligent Automation**. [S.l.], 2019. Citado nas páginas 23 e 39.

TAVARES, J. J.-P. Z. d. S.; SOUZA, G. d. A. Pnrd and ipnrd integration assisting adaptive control in block world domain. **Inproceedings of the International Workshop on Petri Nets and Software Engineering 2019**, p. 73–90, 2019. Citado na página 13.

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. Citado nas páginas 24, 25, 47 e 57.

VAQUERO, T. S. **Post-Design Analysis for AI Planning Applications**. Tese (Doutorado) — Polytechnic School of the University of São Paulo, 2010. Citado na página 77.

VUKSANOVIC, D.; UGARAK, J.; KORCOK, D. Industry 4.0: the future concepts and new visions of factory of the future development. **International Scientific Conference on ICT and E-Business related research**, 2016. Citado na página 13.

WESSTEIN, E. W. **Euler Angles**. 2020. <https://mathworld.wolfram.com/EulerAngles.html>. From MathWorld–A Wolfram Web Resource. Accessed in 28-09-2020. Citado na página 30.

XUE, Z. et al. An automatic grasp planning system for service robots. **International Conference on Advanced Robotics (ICAR)**, 2009. Citado na página 13.

Yi-Liang Chen; Feng Lin. Modeling of discrete event systems using finite state machines with parameters. In: **Proceedings of the 2000. IEEE International Conference on Control Applications. Conference Proceedings (Cat. No.00CH37162)**. [S.l.: s.n.], 2000. p. 941–946. Citado na página 23.

YURISH, S. **Advances in Robotics and Automatic Control: Reviews, Vol. 1, Book Series**. [S.l.: s.n.], 2018. ISBN 978-84-09-02448-3. Citado na página 52.

ZHANG, A. et al. Robust tracking control of robot manipulators using only joint position measurements. **Mathematical Problems in Engineering**, v. 2013, p. 1–9, 11 2013. Citado na página 51.