UNIVERSIDADE FEDERAL DO MARANHÃO

Programa de Pós-Graduação em Ciência da Computação

Diego de Oliveira Dantas

Implementação de um Sistema Autônomo de Construção de Estrutura usando Aprendizado por Reforço

São Luís
2017

Diego de Oliveira Dantas

Implementação de um Sistema Autônomo de Construção de Estrutura usando Aprendizado por Reforço

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal do Maranhão, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.

Universidade Federal do Maranhão - UFMA

Programa de Pós-Graduação em Ciências da Computação

Orientador: Prof. Dr. Areolino de Almeida Neto

Coorientador: Prof. Dr. Sérgio Ronaldo Barros dos Santos

São Luis - MA

2017

Ficha gerada por meio do SIGAA/Biblioteca com dados fornecidos pelo(a) autor(a).
Núcleo Integrado de Bibliotecas/UFMA

de Oliveira Dantas, Diego.
Implementação de um Sistema Autônomo de Construção de Estrutura usando Aprendizado por Reforço / Diego de Oliveira Dantas. – 2017.
115 f.

Coorientador(a): Sérgio Ronaldo Barros dos Santos.
Orientador(a): Areolino de Almeida Neto.
Dissertação (Mestrado) – Programa de Pós-graduação em Ciência da Computação/ccet, Universidade Federal do Maranhão, São Luís, 2017.

1. Aprendizado por reforço. 2. Construção autônoma. 3. Learning automata. 4. Robótica. I. de Almeida Neto, Areolino. II. Ronaldo Barros dos Santos, Sérgio. III. Título.

AGRADECIMENTOS

Primeiramente a Deus que iluminou o meu caminho durante essa caminhada.

Aos meus pais e irmãos pelo amor, incentivo e apoio incondicional.

Aos meus orientadores Areolino de Almeida Neto e Sérgio Ronaldo Barros dos Santos, pelo suporte, pelas suas correções e incentivos durante todo o trabalho.

Aos meus amigos, que entenderam e tiveram paciência nos momentos de minha ausência, dedicados ao estudo. E que me ajudaram em várias partes deste trabalho, mesmo que indiretamente.

A minha namorada, por ter vivenciado comigo todo passo a passo deste trabalho.

Aos colegas de laboratório pelo compartilhamento de ideias e companheirismo

Ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal do Maranhão, pela oportunidade desenvolver este trabalho e a CAPES pelo apoio financeiro.

A todos que direta ou indiretamente fizeram parte da minha formação.

RESUMO

Este trabalho apresenta o desenvolvimento e a implementação de um sistema de construção autônomo, no qual utiliza um robô móvel terrestre para construir estruturas tridimensionais a partir de blocos de diferentes tamanhos. Um planejamento de alto nível é proposto para gerar os planos de construção das estruturas. Esse algoritmo é baseado nos métodos de Aprendizado por Reforço, denominados de *Finite Action-Set Learning Automata* (FALA) e *Parameterized Learnig Automata* (PLA). A partir desse planejador, o usuário define os tipos de blocos empregados na construção e a forma final da estrutura.

O planejador de alto nível é usado para resolver os seguintes problemas: 1) Gerar um diagrama ótimo de montagem, que consiste em uma lista de posições, orientações e tipos de blocos, respeitando a forma final especificada pelo usuário. Esse diagrama é gerado considerando a minimização da quantidade de blocos usados e obedecendo as restrições quanto ao posicionamento dos blocos; 2) Gerar um plano ótimo de execução que é usado pelo robô para realizar a tarefa de montagem da estrutura. Esse plano consiste em definir a sequência de procedimentos para a manipulação e para a montagem dos blocos.

As trajetórias usadas para a realização do plano de execução são geradas por um planejador global composto pelo algoritmo *A**. Ao finalizar o planejamento, o planejador global envia uma série de posições para um controlador de rastreamento de trajetória, chamado de *eband local planner*. Esse controlador de trajetória é usado para controlar a base móvel do robô durante sua navegação através do ambiente simulado ou real. O mapeamento do ambiente simulado e real e a localização do robô nesses ambientes é realizada através do algoritmo chamado de *Real-Time Appearance-Based Mapping* (RTAB-Map). O RTAB-Map usa informações de imagem e de odometria das rodas do robô para gerar o mapa e estimar a posição do robô em relação ao sistema de coordenadas global. Os robôs simulado e real utilizam os recursos do *framework* denominado de *Robot Operation System* (ROS). O ROS permite que diferentes aplicações comuniquem-se entre si, mesmo quando executadas em máquinas diferentes.

Para demonstrar a eficiência das soluções obtidas pelo planejador de alto nível são realizados testes simulados e experimentais do sistema de construção autônomo. Durante esses testes são montadas diferentes tipos de estrutura (Torre, Barragem, Estação Espacial e Pirâmide). Os resultados mostram que o método de aprendizado por reforço é capaz de gerar diagramas de montagem e planos de execução (sequência de procedimentos) factíveis para a realização da tarefa em menor tempo possível.

Palavras-chave: Robótica, aprendizado por reforço, *Learning Automata*, construção autônoma.

ABSTRACT

This work presents the development and implementation of an autonomous construction system in which uses a terrestrial mobile robot for constructing three-dimensional structures from blocks of different size. A high level planning is proposed to generate the construction plans of the structures. This algorithm is based on Reinforcement Learning methods called Finite Action-Set Learning Automata (FALA) and Parameterized Learnig Automata (PLA). From this planner, the used types of blocks for the construction and the final composition of the structure is defined by the user.

The high level planner is used to solve the following problems: 1) Generate an optimal assembly diagram, which consists of a list of positions, orientations and kind of blocks final, taken into account the design of the structure defined by the user. The minimal number of blocks and also the restriction of assembly is considered during the generation of the diagram; 2) Generate an optimal execution plan that can be used by the robot to accomplish the task of assembly. This plan is composed by the sequence of procedures for manipulating and assembling blocks.

The trajectories generated by the global planner based on A* algorithm is used to accomplish the execution plan. After completion the execution plan, the global planner sends a series of positions to a path tracking controller, called eband local planner. This tracking controller is used to control the robot during it navigation through simulated or actual environment. The mapping of the simulated and real environments and the location of the robot in the environment is performed using the algorithm called Real-Time Appearance-Based Mapping (RTAB-Map). The RTAB-Map uses image and odometry information to generate the environment also estimate the position of the robot in relation to the global coordinate system. The simulated and actual robots use the framework called Robot Operation System (ROS). The ROS allows the communication between different applications even if they are performed in different machines.

To demonstrate the efficiency of the obtained solutions using the high level planner, simulated and experimental tests of the autonomous construction system are performed. During these tests, different types of structure (tower, containment wall, space station and pyramid) are assembled. The results show that the reinforcement learning method is able to feasible assembly diagrams and execution plans (sequence of procedures) can be used to perform the task in a short period of time.

Keywords: Robotics, reinforcement learning, Learning Automata, autonomous construction.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ETHZ	*Eidgenössische Technische Hochschule Zürich*
SLAM	*Simultaneous Localization And Mapping*
ASP	*Assembly Sequence Planning*
ANN	*Artificial Neural Network*
GA	*Genetic Algorithm*
ACO	*Ant Colony Optimization*
PSO	*Particle Swarm Optimization*
LA	*Learning Automata*
FALA	*Finite Action-Set Learning Automata*
PLA	*Parameterized Learnig Automata*
i.i.d.	*Independente e identicamente distribuída*
LiPo	*Lithium Polymer*
ROS	*Robot Operating System*
RTAB-Map	*Real-Time Appearance-Based Mapping*
SURF	*Speeded up Robust Features*
SIFT	*Scale Invariant Feature Transform*
RANSAC	*Random Sample Consensus*
TORO	*Tree-based network optimizer*
IMU	*Inertial Measurement Unit*
INS	*Inertial Navigation System*
GPS	*Global Positioning System*

LISTA DE SÍMBOLOS

A	Conjunto de ações do *Learning Automata*
a_k	Uma ação do *Learning Automata*
k	Número total de ações do autômato
p	Distribuição de probabilidade
b	Sinal de recompensa
B	Conjunto de todas as possíveis entradas de reforço do autômato
t	Iterações
$\overrightarrow{p}$	Vetor probabilidade
U	Esquema de aprendizagem usado para atualizar o vetor $\overrightarrow{p}$
λ_1	Taxa de recompensa
λ_2	Taxa de penalidade
u	Vetor usado para geração de probabilidade do método PLA
L	Parâmetro usado para manter o vetor u nos limites
K	Parâmetro usado para manter o vetor u nos limites
n	Parâmetro usado para manter o vetor u nos limites
s	Sequência de variáveis aleatórias i.i.d. (independentes e identicamente distribuídas)
σ	Variância
w_1	Velocidade angular da roda dianteira esquerda
w_2	Velocidade angular da roda dianteira direita
w_3	Velocidade angular da roda traseira esquerda
w_4	Velocidade angular da roda traseira direita
l_x	Distância do centro do robô até o centro da roda em relação ao eixo X
l_y	Distância do centro do robô até o centro da roda em relação ao eixo Y
v_x	Velocidade do robô em relação ao eixo X

v_y	Velocidade do robô em relação ao eixo Y
w_x	Velocidade angular do robô em relação ao eixo Z
r	Raio da roda usada no robô
X_r	Coordenada X do centro do robô
Y_r	Coordenada Y do centro do robô
Z_r	Coordenada Z do centro do robô
X_e	Coordenada X do eletroímã
Y_e	Coordenada Y do eletroímã
Z_e	Coordenada Z do eletroímã
X_g	Coordenada X global fixa no ambiente
X_g	Coordenada Y global fixa no ambiente
Z_g	Coordenada Z global fixa no ambiente
θ_{j1}	Ângulo do braço do robô
θ_{j1}	Ângulo da garra do robô
L_{z1}	Altura da junta 1 em relação ao eixo Z_r
L_{z2}	Altura da junta 2 em relação ao eixo Z_r
L_a	Comprimento do braço do robô
x_e	Posição da garra do robô em relação a global
y_e	Posição da garra do robô em relação a global
z_e	Posição da garra do robô em relação a global
θ_e	Orientação da garra do robô em relação a global
x_r	Posição do centro do robô em relação a global
y_r	Posição do centro do robô em relação a global
z_r	Posição do centro do robô em relação a global
θ_r	Orientação do centro do robô em relação a global
m_p	Valor em micropassos configurados no drive a4988

ρ	Resolução do motor de passo
f_m	Frequências de pulso aplicadas pelo *driver* a4988
φ	Dimensão do bloco de uma unidade
M_{ent}	Matriz tridimensional binária que serve de entrada para o algoritmo de geração do Diagrama de Montagem
M_{Xtotal}	Dimensão da matriz tridimensional binária M_{ent}
M_{Ytotal}	Dimensão da matriz tridimensional binária M_{ent}
M_{Ztotal}	Dimensão da matriz tridimensional binária M_{ent}
M_{aut}	Matriz de autômatos
V_b	Conjunto de blocos disponíveis para a montagem
M_X	Valor de posição da matriz
M_Y	Valor de posição da matriz
M_Z	Valor de posição da matriz
M_{Ctotal}	Valor total de camadas concêntricas
M_C	Valor de camadas concêntrica
F_r	Total de falhas de conexão
F_b	Total de blocos usados na estrutura
F_{rb}	Valor da função de custo da geração do diagrama de montagem
J	Memória do autômato, guarda os valores da função de custo
R	Número de valores que a variável J pode armazenar
J_{med}	Média dos R valores de J
J_{min}	Menor valor armazenado em J
X_{offset}	Valor de deslocamento do bloco no ambiente no eixo X
Y_{offset}	Valor de deslocamento do bloco no ambiente no eixo Y
D_A	Diagrama de montagem
I_A	Identificador do bloco no diagrama de montagem

n_d	Número total de elementos do diagrama de montagem
L_A	Lista dos locais de manipulação
I_M	Identificador do bloco da lista dos locais de manipulação
F_a	Deslocamento total do robô no procedimento de montagem
F_m	Deslocamento total do robô no procedimento de manipulação
F_{ma}	Valor da função de custo da geração do plano de execução
n_i	Número total de elementos do plano de execução

SUMÁRIO

1 INTRODUÇÃO

A construção autônoma é um campo interessante que concentra-se na aplicação de métodos de controle de sistemas e de Inteligência Artificial em plataformas robóticas, nas quais são usados para realizar tarefas de montagem. Em outras palavras, trata-se em aplicar as mais recentes tecnologias de automação nas áreas de construção de estruturas civis (como barragens, pontes, edifícios, torre, gasodutos etc.) e também em construções militares e aeroespacial.

A construção autônoma surge para diminuição de acidentes de trabalho e redução na duração do processo de construção. Além dos aspectos mencionados, os robôs poderão potencialmente realizar tarefas de construção, onde a presença humana é impossível, indesejável ou insegura (WAWERLA; SUKHATME; MATARIC, 2002). Por exemplo, a construção em áreas perigosas após desastres naturais ou causados pelo homem, como terremotos e acidentes nucleares, a construção em ambientes submarinos ou espaciais, a construção em áreas que não são facilmente acessíveis aos seres humanos e a construção onde uma estrutura inicial é necessária para preparar para um habitat humano.

1.1 MOTIVAÇÃO

A área de construção ainda não está preparada para o uso de robôs em suas tarefas. A maioria das tarefas de construção ainda são executados por humanos. Isso não é surpreendente considerando o ambiente dinâmico, por exemplo, da construção de um prédio. Ambientes em constante mudança, como terrenos de difícil locomoção ou não estruturado são alguns dos maiores desafios a serem superados, antes que um robô totalmente autônomo possa ser usado.

Esses desafios tornam a construção autônoma uma tarefa interessante para a pesquisa. A tarefa de construção envolve problemas de diferentes áreas de pesquisa dentro do domínio da robótica. Entre tais problemas pode-se destacar a localização e o mapeamento de ambientes desconhecidos, o controle de navegação, o planejamento da tarefa e a execução das ações estabelecidas para o robô. Além de ser um tema de pesquisa interessante, a construção autônoma pode ter várias aplicações no mundo real no futuro.

Apesar do progresso significativo nos processos de construção e nos procedimentos de segurança, os procedimentos de montagem podem ser cansativos e muitos perigosos para os seres humanos. Portanto, é desejável utilizar os robôs para executar as tarefas repetitivas e em ambientes hostis. Isso pode ser útil para a exploração de locais com um ambiente severo, como lugares subaquáticos, regiões polares do planeta ou até mesmo no espaço ou em outros planetas.

A pesquisa na área da construção autônoma é, portanto, uma maneira de aliviar a vida cotidiana dos trabalhadores e facilitar o desenvolvimento de infraestruturas em lugares ainda não explorados pelos humanos.

Neste trabalho é apresentado uma abordagem na qual utiliza o aprendizado por reforço

para o planejamento da construção autônoma de uma estrutura de três dimensões, especificada pelo usuário. O sistema de montagem autônomo é dividido em três etapas:

1. Gerar um diagrama ótimo de montagem, que consiste em uma lista de posições, orientações e tipos de blocos, respeitando a forma final especificada pelo usuário. Esse diagrama é gerado considerando a minimização da quantidade de blocos usados e obedecendo as restrições quanto ao posicionamento dos blocos;

2. A geração de um plano ótimo de montagem. Nessa etapa o algoritmo deve conhecer a localização inicial dos blocos disponíveis no ambiente e a posição dos blocos na estrutura final. A primeira já é conhecida e a segunda é gerada pelo algoritmo de montagem. O algoritmo então, gera uma sequência de manipulação dos blocos que é realizada pelo robô, e também, uma sequência de montagem da estrutura que especifica a forma que os blocos serão posicionados pelo robô. Essas sequências devem resultar em uma montagem de menor tempo possível, levando em consideração geometria do robô e dos blocos;

3. Execução da montagem com o uso de um robô móvel terrestre. Essa etapa o robô recebe as informações das posições iniciais e finais que deve ir, a partir delas ele gera um conjunto de procedimentos para realizar essa tarefa. Usando tais procedimentos, o robô terrestre é capaz de manipular, transportar e montar um conjunto de diferentes blocos, a partir do plano de construção.

Todas as etapas mencionadas são executadas sem a intervenção do usuário.

1.2 OBJETIVOS

O objetivo principal deste trabalho é desenvolver um sistema de montagem autônomo usando um robô móvel terrestre omnidirecional. Um algoritmo é proposto para a geração do plano de montagem de diferentes estruturas usando um método de aprendizado por reforço conhecido como *Learning Automata*.

Como o sistema de montagem autônomo integra alguns subsistemas fundamentais, os objetivos específicos são:

1. Implementar o algoritmo *Finite Action-Set Learning Automata* (FALA) e *Parameterized Learnig Automata* (PLA) para a geração do diagrama de montagem da estrutura;

2. Implementar o algoritmo PLA para geração do plano de execução da estrutura;

3. Desenvolver o robô móvel terrestre omnidirecional constituído por um braço com um eletroímã em seu terminal;

4. Desenvolver o ambiente de simulação baseado no simulador V-REP;

5. Desenvolver o modelo computacional do robô usado;
6. Realizar a simulação da montagem das estruturas;
7. Realizar as validações experimentais do sistema de mapeamento e localização e da montagem de uma estrutura.

1.3 ORGANIZAÇÃO DO TRABALHO

Esta dissertação está organizada da seguinte maneira:

- Capítulo 1: Este capítulo tem um caráter introdutório e destina-se a orientar o leitor sobre o problema abordado na dissertação, bem como a motivação para o seu estudo, os objetivos a serem alcançados e uma descrição da organização da dissertação;
- Capítulo 2: Esse capítulo começa com o estado da arte relativo às tecnologias atualmente existentes na área de robôs construtores. Em seguida são apresentados os diversos modelos de câmeras de profundidade. Por fim, é finalizado com uma breve introdução aos métodos de aprendizado por reforço usado nesta dissertação;
- Capítulo 3: Esse capítulo apresenta os recursos de *hardware* e *software* adotados no projeto. São apresentados os componentes embarcados, tais como o microprocessador e os sensores, além do computador de bordo e o sistema de potência. Em seguida são apresentadas as equações cinemáticas do robô usado. Além disso são explicados o funcionamento dos algoritmos utilizados, o software instalado no computador e o ambiente de simulação;
- Capítulo 4: Nesse capítulo são abordados os materiais usados na confecção dos blocos de montagem, a classificação quanto as suas dimensões e o local de armazenagem em relação a uma coordenada global. Também são apresentados os tipos de estruturas a serem construídas e o ambiente usado para a montagem, tanto no modelo simulado quanto no modelo real;
- Capítulo 5: Nesse capítulo é apresentada à metodologia utilizada para a geração do diagrama de construção e do plano de montagem das estruturas;
- Capítulo 6: Nesse capítulo são apresentados os resultados dos algoritmos de construção de estruturas. O capítulo explica a melhor maneira de escolher os parâmetros para geração da estrutura. No ambiente simulado foi realizado a montagem de quatro diferentes estruturas usando o diagrama ótimo de montagem e o plano ótimo de execução. No final foi feito a validação do sistema de construção da estrutura em um ambiente real;
- Capítulo 7: Nesse capítulo são apresentadas as conclusões e as propostas para os trabalhos futuros.

2 ESTADO DA ARTE

A pesquisa em robótica aplicada à construção começou na década de 1980, e desde então o desenvolvimento desses sistemas têm levado a criação de uma ampla gama de plataformas robóticas. Devido a esta diversidade, foram consideradas várias categorias gerais de robôs construtores (SAIDI; O'BRIEN; LYTLE, 2008). A primeira consiste em sistemas teleoperados, nos quais as máquinas estão sob controle remoto de seres humanos. Um operador humano interpreta as situações do robô e elabora um plano para executar as tarefas, transmitindo ordens que são transformadas em ações pelo robô.

A segunda categoria é denominada de máquinas de construção programáveis. Essas máquinas necessitam que o operador humano definam as subtarefas a serem executadas a partir de uma lista de funções pré-programadas. Note que uma nova função pode ser programada pelo operador.

A terceira categoria consiste em sistemas inteligentes. Os robôs de construção não tripulados executam suas tarefas em um modo semi ou totalmente autônomo. No modo de construção semiautônomo, um robô realiza suas tarefas com algum nível de autonomia feito através da interação com um supervisor humano. Em contraste, no modo totalmente autônomo, os robôs são capazes de completar as tarefas sem intervenção humana dentro de um domínio específico.

Este capítulo é restrito a sistema de construção usando robôs móveis autônomos (ou semiautônomos).

Este capítulo apresenta uma breve descrição sobre câmeras de profundidade e seus modelos disponíveis. Além de uma introdução ao método de aprendizagem por reforço *Learning Automata*, que é um método estocástico em que cada autômato deve maximizar sua recompensa de acordo com a resposta do ambiente e as ações escolhidas.

2.1 ROBÔS CONSTRUTORES

Desenvolvimentos recentes em sistemas robóticos levaram a um aumento do uso de robôs aéreos ou terrestres em aplicações como a construção de alvenaria, a construção de estruturas treliçadas, a construção de móveis, dentre outras (ARDINY; WITWICKI; MONDADA, 2015). De certa maneira, uma construção completa consiste em um número finito de tarefas, tais como manuseio de material, o transporte, a alocação e a fixação. Um robô pode executar uma ou mais destas tarefas dependendo da situação e de sua capacidade de locomoção e de manipulação. Existem alguns robôs já sendo usados em pesquisa para a construção. Eles podem ser classificados dependendo do material que manipulam, do tipo de mobilidade usada para locomover ou do método usado para a construção das estruturas. Nas subseções a seguir são mencionados alguns estudos de acordo com esses critérios.

2.1.1 Quanto ao material

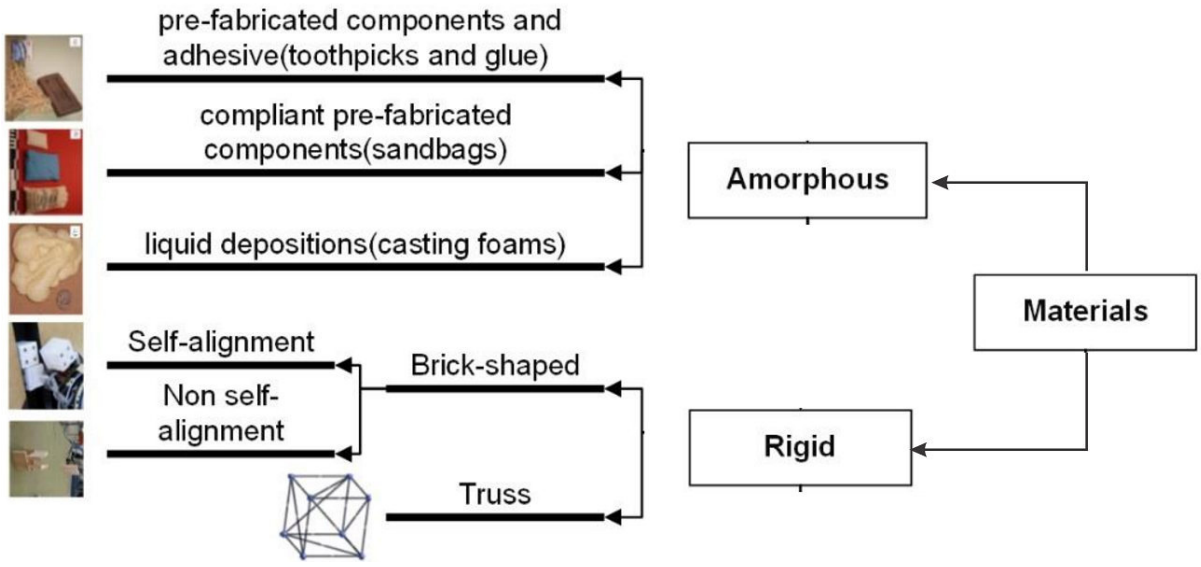
Uma maneira de classificar robôs construtores é quanto ao material usado por eles na construção (ARDINY; WITWICKI; MONDADA, 2015).

As propriedades dos materiais usados devem ser levadas em consideração, porque o tipo de material determina qual tipo de robô pode ser melhor aplicado no processo de construção. Note que fatores como a forma da estrutura a ser construída, a precisão da construção, a velocidade de construção, a simplicidade e a quantidade de material usado, também possuem um forte impacto no projeto construtivo do robô.

Na natureza observa-se facilmente animais ou insetos construindo seus ninhos com materiais disponíveis ao seu redor. Percebe-se que pássaros usam galhos e folhas para montar seus ninhos em árvores ou em outros locais, sem que haja nenhuma espécie de cola para fazer conexão entre os galhos. Sendo que esse trabalho é feito apenas a partir do encaixe dos materiais. As formigas já possuem um método diferente, sendo que no lugar da adição de material elas fazem a subtração deles. Nesse caso é a retirada de terra do solo para a construção de túneis. Outros exemplos são os cupins que usam uma espécie de massa a partir de uma mistura de solo, de partículas de madeira, de saliva e fezes, para construir seus ninhos. Por sua vez, as construções para o uso dos seres humanos são geralmente mais complexas, sendo que também precisam de materiais mais complexos.

A Figura 1 mostra a classificação dos robôs construtores em função dos materiais usados e do ambiente alvo para a construção das estruturas. Um robô, por exemplo, que precise ejetar espuma deve possuir um efetuador diferente em relação a um robô que agarre blocos.

Figura 1 – Classificação dos materiais utilizados na construção autônoma.



Fonte: Ardiny, Witwicki e Mondada (2015)

Três tipos de materiais amorfos [1] para construção foram estudados em Napp *et al.* (2012): componentes rígidos com adesivos (palitos e cola) na Figura 2a, componentes pré-fabricados (sacos de areia) na Figura 2b e materiais líquidos que tornam-se sólidos (espumas) na Figura 2c. Nessa pesquisa, o alto índice de expansão da espuma utilizada foi um ponto atraente, mas foi necessário esperar um tempo suficiente para curar a espuma. Os objetos feitos com os palitos e a cola possuem características intermediária, que é a alta expansão e o baixo tempo de cura. Os sacos de areia não precisaram de um mecanismo complexo para o transporte, mas não criaram estruturas permanentes.

Figura 2 – Rampas construídas a partir de três tipos de materiais amorfos.

Fonte: Napp *et al.* (2012)

A pesquisa sobre o uso de materiais amorfos tem como alvo principalmente a fabricação digital (Produção de objetos físicos, a partir de modelos digitais, o método mais conhecido é chamado de impressão 3D), considerando a deposição ou remoção contínuas. Em Gershenfeld *et al.* (2015) abordaram as implicações desse tipo de material na fabricação digital. Uma aplicação é a construção de contornos desenvolvidos para a construção de uma grande estrutura em uma única operação (KHOSHNEVIS *et al.*, 2005). Os avanços nos sistemas robóticos aplicados à fabricação digital de grandes estruturas permitiram aos arquitetos construírem estruturas elegantes. A fabricação digital pretende preencher as lacunas entre as tecnologias digitais e o processo de construção física, pois as restrições de projeto podem ser flexibilizadas permitindo que as estruturas sejam fabricadas com alta personalização e sofisticação.

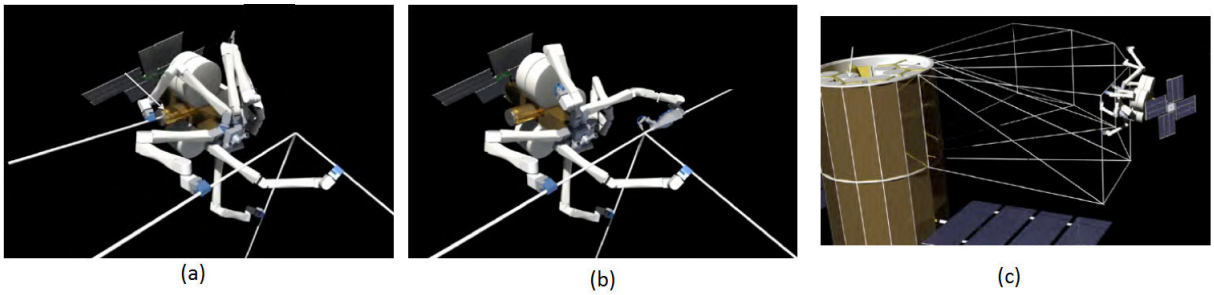
Em aplicações espaciais, os processos de fabricação digital podem ser úteis em situações onde as agências espaciais podem lançar somente a matéria-prima ao espaço, com o objetivo de reduzir o volume transportado. O volume, a massa e o custo são fatores significativos quando se trata de aplicações no espaço. Então para garantir missões bem-sucedidas, a diminuição de tamanho e da massa dos objetos enviados (usados para construção) é muito importante, principalmente, em sistemas espaciais de grande porte, como antenas ou painéis.

O *SpiderFab* é usado para empregar técnicas semelhantes a impressão 3d para fabricar componentes em órbita, permitindo a NASA escapar das limitações volumétricas do lançamento (HOYT, 2013). O *SpiderFab* (HOYT, 2013) é equipado com um *Extruder Spinneret* usado para

[1] Que não tem forma determinada (AURELIO, 2017).

converter carretéis de fio enrolados em tubos compósitos[2] de alta performance, como ilustrado na Figura 3a. Esse sistema usa uma ferramenta de alta precisão *Joiner Spinneret* que adapta técnicas de impressão 3D para criar conexões de alta resistência entre os elementos estruturais, como ilustrado na Figura 3b. Note que o *SpiderFab* é capaz de construir estruturas grandes e esparsas. A Figura 3c ilustra o *SpiderFab* construindo uma estrutura de suporte para uma antena em um barramento de um satélite.

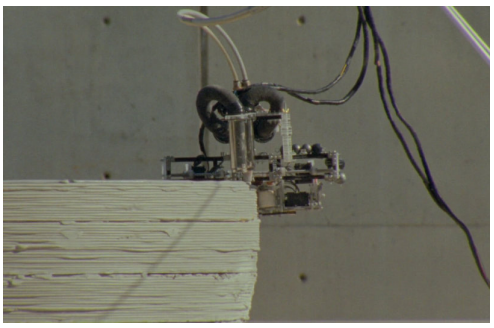
Figura 3 – Robô *SpiderFab* na criação dos elementos estruturais em (a), imprimindo conexões em (b) e em (c) construindo uma estrutura de suporte para o satélite.



Fonte: Hoyt (2013)

Em Jokic *et al.* (2014), os autores usaram o robô Grip Robot (Figura 4) que se prende a estrutura que está sendo impressa ao mesmo tempo que ele se move ao longo dela. Esse robô pode cobrir uma área muito maior do que o seu próprio tamanho, tornando-se independente da dimensão do objeto a ser impresso.

Figura 4 – *Grip Robot* construindo uma estrutura enquanto movimenta-se.



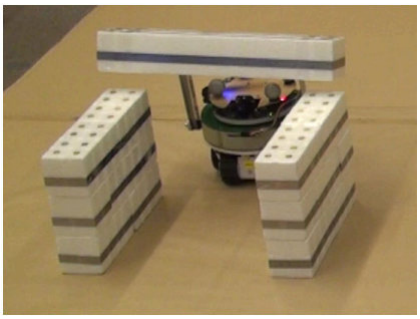
Fonte: Markopoulou (2017)

A construção autônoma é um processo complexo no qual muitas falhas podem ocorrer. Essas falhas podem propagar-se de uma etapa para outra, por exemplo, se um robô captura um bloco e, em seguida, colocar esse bloco em um local próximo, mas não no local exato como definido pelo diagrama da estrutura, isso pode comprometer as etapas seguintes causando até

[2] Materiais com combinações incomuns que não podem ser atendidas pelas ligas metálicas, cerâmicas e polímeros convencionais. Exemplo: Fibra de Carbono.

uma destruição da estrutura já construída. Portanto, é importante evitar ou corrigir as falhas como essas. Uma maneira de evitar isso é usar peças de material rígido com capacidade de autoalinhamento que podem minimizar ou retirar os erros de desalinhamento. Por exemplo, blocos feitos de poliestireno, e compostos por ímãs para a realização do autoalinhamento foi usado em Wismer *et al.* (2012). Esse trabalho apresenta uma estrutura composta por blocos de poliestireno equipados com pequenos ímãs nas suas faces superiores e inferiores, para que ao empilhar uma peça na outra, os ímãs possam fazer o alinhamento entre elas. Nesta pesquisa foram utilizados blocos de três diferentes tamanhos para a construção da estrutura (Figura 5).

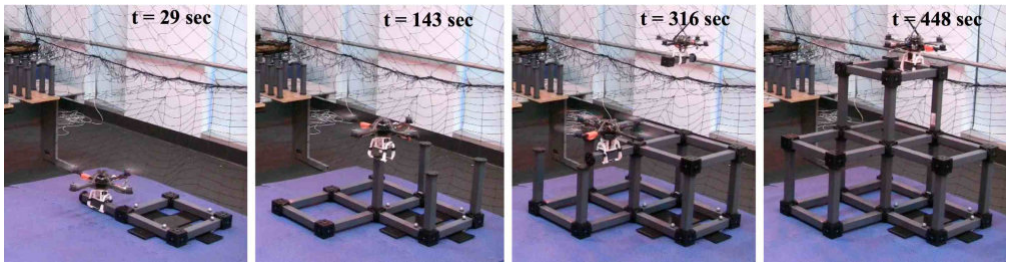
Figura 5 – Estrutura com blocos de três tamanhos diferentes autoalinhados através de ímãs.



Fonte: Wismer *et al.* (2012)

Outro uso para materiais rígidos é a construção de estruturas treliçadas, formadas por vigas, colunas e conexões (LINDSEY; MELLINGER; KUMAR, 2012; SANTOS; GIVIGI; NASCIMENTO, 2015). As vigas e colunas podem ser ligadas em conjunto para criar uma estrutura cúbica simples. Dessa forma, pode-se construir vários cubos unidos uns com os outros. No final do processo, a junção entre os cubos define o formato da estrutura que pode ser semelhante a uma torre se os cubos forem construídos um sobre os outros. Em Lindsey, Mellinger e Kumar (2012) um time de quadrirrotores manipulam vigas e colunas que possuem uma conexão fêmea de um lado e conexão macho do outro. Essas conexões possuem ímãs que proporcionam uma ligação de autoalinhamento (Figura 6).

Figura 6 – Fotos tiradas ao decorrer do tempo na construção de uma pirâmide.

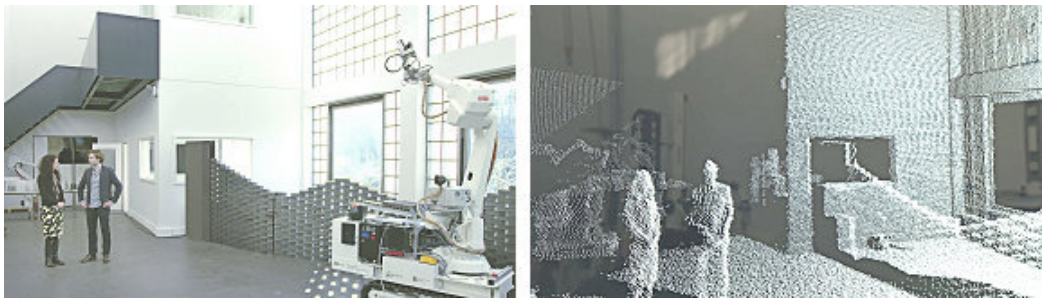


Fonte: Lindsey, Mellinger e Kumar (2012)

Para peças que não possuem mecanismos de autoalinhamento, sistemas robóticos avançados são necessários para atender aos requisitos de construção. Em Helm *et al.* (2012) é

apresentado o *DimRob* (Figura 7), que é equipado com um manipulador e um *scanner* a laser 3D que é usado para digitalizar os tijolos de madeira colocados durante a fabricação, em seguida, envia essa medida mapeada para o software do controlador para obter os próximos comandos.

Figura 7 – Robô *DimRob*.



Fonte: Dorfler (2017)

2.1.2 Quanto à mobilidade

Nesse campo, os robôs construtores são tipicamente divididos em robôs aéreos e robôs terrestres. Os robôs aéreos, como quadrirrotores, necessitam de um sistema de posicionamento preciso para serem usados na construção. Nesse caso um sistema de localização externo é empregado para fornecer um voo de alta precisão para tarefas de construção (SANTOS; GIVIGI; NASCIMENTO, 2015).

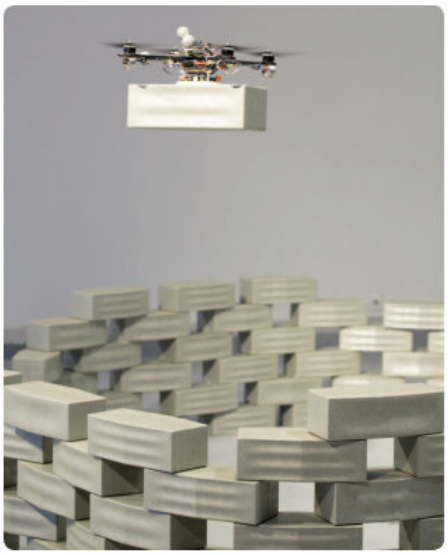
Os robôs aéreos voam até o ponto de construção e colocam os blocos diretamente na posição desejada, sem precisar escalar andaimes para chegar em altitudes elevadas. Diferentes estruturas podem ser construídas, uma vez que os quadrirrotores podem mover-se no espaço 3D. Portanto, eles podem colocar e manipular o material de acordo com um plano predeterminado. Por outro lado, no momento, a maioria dos robôs aéreos têm capacidade de carga útil limitada, mas vários robôs aéreos conseguem agarrar e transportar um objeto pesado cooperativamente, de acordo com Mohammadi *et al.* (2016).

Outra limitação diz respeito às considerações aerodinâmicas porque a forma das peças manipuladas pode afetar o desempenho do controle e da estabilidade. Dessa forma, as peças de construção devem ser desenvolvidas de modo a satisfazer a restrição aerodinâmica. Além disso, projetar um sistema de controle de robôs aéreos com distúrbios significativos (por exemplo, rajadas de vento e variação no peso da carga) não é uma tarefa fácil (CABRAL *et al.*, 2017; POUNDS; BERSAK; DOLLAR, 2012).

Em um experimento realizado pelo ETH Zurique (Instituto Federal de Zurique), quatro quadrirrotores foram usados para construir uma torre de tijolos (Figura 8) (WILLMANN *et al.*, 2012). A pose dos robôs foi assegurada por um sistema de câmeras operando em tempo real. A orientação de referência para os robôs é determinada de acordo com um desenho digital, permitindo a captura e o depósito dos blocos pelo robô.

Nesse trabalho, o sistema de rastreamento de movimento da VICON, que utiliza câmeras infravermelhas, foi usado para estimar a posição e a orientação dos objetos e dos robôs aéreos. Fornecendo um *feedback* de posição a 150 Hz com precisão na ordem de um milímetro.

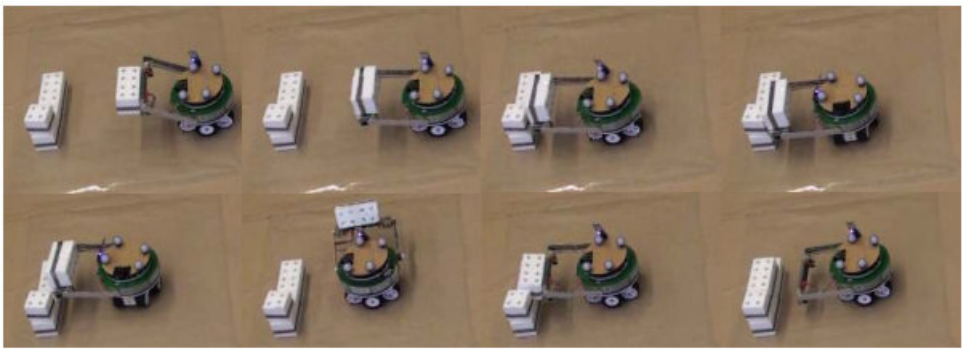
Figura 8 – Montagem de um bloco.



Fonte: Willmann *et al.* (2012)

Em contraste com robôs aéreos, os robôs terrestres são mais estáveis e de fácil controle. Além disso, eles podem transportar objetos mais pesados e de maior complexidade em termos de forma, embora dificilmente consigam acessar certos espaços, que podem ser acessados facilmente pelos robôs aéreos. Na pesquisa de Magnenat, Philippsen e Mondada (2012) foi usado o robô *marXbot* para capturar blocos com capacidade de autoalinhamento (Figura 9). Eles empregaram odometria, além de uma câmera frontal e sensores de proximidade para fornecer as informações necessárias para captura e soltar os blocos. Uma extensão desse trabalho foi desenvolvida para construir uma estrutura coberta. Nessa tarefa, eles usaram um sistema de câmeras VICON para estimar a posição do *marXbot* (WISMER *et al.*, 2012).

Figura 9 – Montagem dos blocos no segundo nível da construção de uma parede.



Fonte: Wismer *et al.* (2012)

Em Knepper *et al.* (2013) é apresentado o *IkeaBot*, um robô móvel com um manipulador em sua base. Nesse trabalho, uma equipe de *IkeaBot* é usada para a montagem de mobílias, como por exemplo, mesas e cadeiras. Esses robôs são capazes de localizar peças no seu espaço de trabalho e realizar operações como aparafusamento para conectar as peças. Porém, para isso foi preciso desenvolver um efetuador capaz de executar uma manobra natural de aparafusamento. O efetuador gera um movimento giratório contínuo em componentes de vários tamanhos. Durante o processo de montagem, um dos robôs deve colocar e manter a peça e outro robô parafusar e finalizar a montagem.

A Figura 10 mostra dois robôs que trabalham para aparafusar uma perna no tampo de uma mesa. O algoritmo de planejamento é executado *on-board* e de forma distribuída. O *IkeaBot* é baseado no KUKA *youbot*, o qual é constituído por rodas omnidirecionais do tipo sueca.

Figura 10 – Dois robôs colaborando na montagem de uma mesa.

Fonte: Knepper *et al.* (2013)

2.1.3 Quanto ao algoritmo de construção

A natureza mostra que insetos sociais como formigas, abelhas, vespas e cupins podem construir estruturas complexas que são de ordem de grandeza maior do que os próprios insetos. Já sabe-se que esses animais dependem fortemente de um conceito chamado *Stigmergy* (KARSAI; PÉNZES, 1993). *Stigmergy* é um mecanismo que permite a comunicação indireta, e a coordenação entre os agentes que trabalham juntos em uma tarefa. Ao deixar um traço no ambiente (como por exemplo, uma peça da estrutura), um determinado agente estimula a execução de certas ações em outros agentes (como concluir a estrutura).

Várias pesquisas foram feitas analisando o comportamento de enxames e no *stigmergy*. Uma dessas pesquisas mostra que um enxame de robôs com capacidade muito limitadas de detecção e cognição podem construir estruturas parecidas a uma parede, enquanto ele é governado por um pequeno conjunto de regras muito simples (STEWART; RUSSELL, 2006).

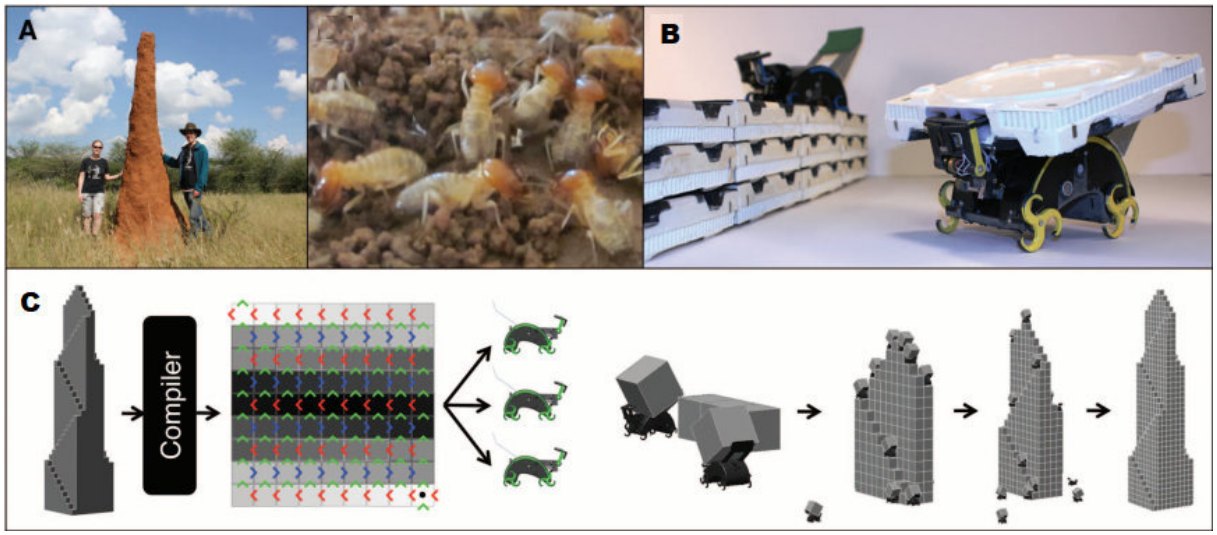
Os trabalhos bioinspirados geralmente não são conduzidos por uma potencial aplicação no mundo real, mas pretendem melhorar a compreensão do comportamento do enxame e os

algoritmos de construção, já que eles estão presentes na natureza. Portanto, esses sistemas geralmente não são projetados para construir estruturas perfeitas e reprodutíveis, mas estruturas aproximadas.

Abordagens bioinspiradas são elegantes, pois robôs móveis simples são capazes de executar a construção automatizada seguindo regras e executando algoritmos reativos. Cada indivíduo age de forma independente, onde a interação entre eles e o ambiente garante a construção automatizada sem um modelo convencional. A abordagem bioinspirada pode ser mais robusta à falha por causa de seus métodos descentralizados, que podem ser flexíveis e até mesmo incluir mecanismos de autoreparo.

Werfel, Petersen e Nagpal (2014) apresentaram um sistema de robôs móveis terrestres (Figura 11b) capaz de realizar a construção autônoma inspirada nas atividades desempenhadas pelos cupins (Figura 11a). Os robôs são capazes de escalar a estrutura durante a construção, na qual é composta por blocos rígidos. O objetivo desta pesquisa é usar os princípios dos insetos para construção de uma estrutura definida pelo usuário para fins civis. A partir de um modelo definido pelo usuário, um compilador *off-line* gera regras de tráfego para os robôs realizarem a montagem (Figura 11c).

Figura 11 – Robôs TERMES manipulando os blocos.



Fonte: Werfel, Petersen e Nagpal (2014)

De fato, os princípios de construção bioinspirados e a arquitetura humana têm abordagens fundamentalmente diferentes. Os seres humanos constroem estruturas baseadas em um projeto, e os processos de construção são centralizados a partir de um plano de construção. Para seguir essa abordagem, os robôs devem ter uma representação global do ambiente para poder construir uma estrutura baseada em planos pré-especificados, como acontece, por exemplo, com os robôs *MarXBot*, *IkeaBot* e o *DimRob*. Esses robôs seguem uma lista de instruções de posicionamento, sequenciadas por ordem de colocação denominada de plano de montagem.

2.1.4 Quanto aos algoritmos de planejamento de sequência de montagem

O problema do Planejamento de Sequência de Montagem (ASP) preocupa-se em encontrar uma sequência de operações que não resulte em inconsistência durante a montagem (como, por exemplo, monta uma peça em nível superior sem que o nível inferior esteja montado). Em geral, o ASP é gerado através da geometria final da estrutura e das posições de montagem de cada peça (JIMÉNEZ, 2013). Uma visão geral do ASP é apresentada em (JIMÉNEZ, 2013), em sua pesquisa foi incluída os elementos de um planejamento de sequência:

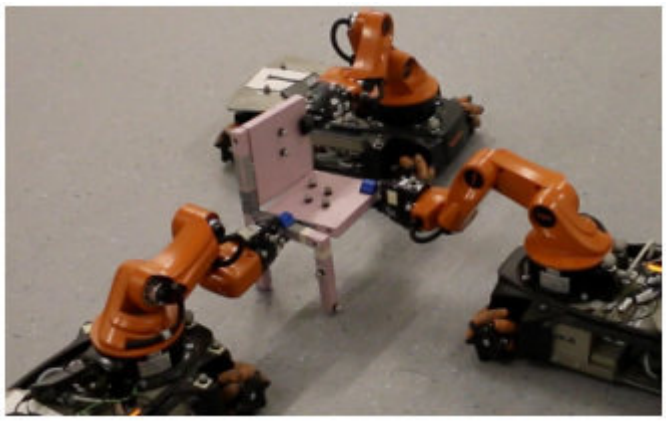
- Encontrar uma sequência de montagem ótima de acordo com um ou mais critérios operacionais;
- Representar o espaço de sequências de montagem;
- Aplicar algoritmos de busca ou otimização satisfazendo as restrições existentes entre subconjuntos de montagem.

Atualmente, a pesquisa no planejamento da sequência de montagem pode ser agrupada em duas categorias principais: métodos de busca exaustiva, baseadas em grafos e métodos baseados em inteligência artificial. A fim de facilitar a representação e geração de planos de montagem, as estruturas podem ser modeladas como grafo ou grafos AND/OR. Os métodos de busca exaustiva geralmente utilizam um algoritmo de poda ou A* para realizar uma pesquisa no grafo e encontrar as soluções ótimas ou até mesmo globais (MELLO; SANDERSON, 1991). No entanto, o número de soluções potenciais aumenta exponencialmente com o aumento do número de peças, o que significa que uma explosão combinatória é inevitável e é impraticável.

Sabe-se que o ASP é um problema combinatório de grande escala e por isso bem limitado, sendo muito difícil aplicar técnicas convencionais para solução e otimização. Nas duas últimas décadas, muitas técnicas de computação baseadas em inteligência artificial foram utilizadas, tais como, as Redes Neurais (ANN) (SINANOGLU; RIZA, 2006), os Algoritmos Genéticos (GA) (PEDRAZA; DIAZ; LOMBERA, 2016), a Otimização por Colônias de Formigas (ACO) (SHARMA *et al.*, 2009), a Otimização de Enxames de Partículas (PSO) (IBRAHIM *et al.*, 2016) e entre outros.

Na pesquisa de Wei (2012), foi aplicada uma geração de sequência de montagem automática para construção de navios, considerando os tamanhos e as posições. Em Dobashi *et al.* (2014), foram considerados os espaços livres de colisão entre os objetos manipulados e os objetos montados durante a construção, embora a sequência de montagem seja predefinida manualmente considerando essas restrições. McEvoy, Komendera e Correll (2014) consideraram a estabilidade e as restrições geométricas no planejamento das sequências de montagem das estruturas do tipo treliça. Em Dogar *et al.* (2015), foram usados vários robôs móveis para montar uma cadeira, como mostra a Figura 12.

Figura 12 – Robôs montando uma cadeira.



Fonte: Dogar *et al.* (2015)

2.2 CÂMERAS DE PROFUNDIDADE

Uma câmera de profundidade pode ser descrita como uma câmera de cor regular com a capacidade de criar imagens com dados de profundidade. A câmera de profundidade pode ser mais precisamente descrita como uma câmera RGB-D. A imagem RGB-D é composta por canais de cores e informações geométricas da cena, onde cada pixel da imagem capturada é composta por uma combinação de três canais de cores (um para cada cor primária: vermelho, verde e azul), e também as informações geométricas que são os dados de profundidade dos pixels (CRUZ; LUCIO; VELHO, 2012).

As técnicas de captura de dados geométricos baseadas em imagem podem ser divididas em duas categorias: estéreo passivo e estéreo ativo. De maneira semelhante ao olho humano, as técnicas de etéreo passivo utilizam duas ou mais câmeras calibradas, onde são usadas para calcular as correspondências entre as projeções dos pontos em imagens distintas, medindo a profundidade de cada ponto na cena. A técnica de estéreo ativo tem como base os mesmos princípios da estereoscópica passiva, entretanto utilizam um par de câmeras, que ilumina a cena com um padrão de luz estruturada.

O Microsoft Kinect foi um dos primeiros dispositivos de fácil acesso a fornecer imagens RGB-D (CRUZ; LUCIO; VELHO, 2012). O sucesso nas vendas desse dispositivo, tanto para entretenimento, quanto para o uso em pesquisa, provavelmente, motivou o desenvolvimento de outros dispositivos, como por exemplo o Asus Xtion PRO LIVE (ASUS, 2017), o *Project Tango* do Google (TANGO, 2017), o Realsense da Intel (SENSE, 2017), *Structure Sensor* da Occipital (OCCIPITAL, 2017), dentre outros.

As câmeras RGB-D baseadas na tecnologia desenvolvidas pela PrimeSense, como o Microsoft Kinect e o Asus Xtion PRO LIVE, foram usadas para odometria visual em um quadrirrotor (HUANG *et al.*, 2016), reconstrução 3D (NEWCOMBE *et al.*, 2011) e Mapeamento e Localização Simultânea (SLAM) (OLIVER *et al.*, 2012). O uso dessas câmeras deve-se ao seu

baixo consumo de energia e ao seu leve peso em comparação com outras câmeras estéreo. Outra vantagem é que elas executam o cálculo de profundidade na própria câmera.

2.2.1 Microsoft Kinect

O Microsoft Kinect (Figura 13) baseia-se na tecnologia desenvolvida pela PrimeSense e é composto por uma câmera RGB (VGA 640x480, CMOS), um sensor de profundidade que permite visualizar o ambiente a sua volta em 3 dimensões, um microfone, um acelerômetro em 3 dimensões, e um motor na base para alterar a orientação das câmaras (ZHANG, 2012). A combinação do sensor de profundidade com a câmera RGB refere-se ao sensor RGB-D.

Figura 13 – Microsoft Kinect.

Fonte: Kinect (2017)

O sensor de profundidade consiste em uma fonte infravermelho e em um laser que projeta um padrão de pontos que é recebido por um sensor CMOS monocromático (ZHANG, 2012). Esta câmera infravermelha recebe o padrão refletido e converte intensidades em distâncias. A sensibilidade do Microsoft Kinect não depende em nada da luminosidade do ambiente. Ele pode trabalhar perfeitamente em um ambiente sem luz. A gama de valores medidos pelo Microsoft Kinect não é linear com a distância. As distâncias detectáveis estão entre os 0,45 e 6 m com resolução de 1 centímetro no eixo Z, enquanto nos eixos X e Y a resolução é da ordem de milímetros. O *framerate* da saída é de 30 Hz.

A capacidade de percepção de profundidade do Microsoft Kinect, combinadas com seu custo relativamente baixo e alta acessibilidade, contribuíram para tornar esse dispositivo muito popular em projetos de pesquisa relacionados a SLAM e robótica (CAI *et al.*, 2016).

2.2.2 Asus Xtion PRO LIVE

O Asus Xtion PRO LIVE (Figura 14) baseia-se na tecnologia desenvolvida pela PrimeSense que fornece imagens RGB e de profundidade, com resolução VGA (640 x 480 pixels) a uma taxa de 30 quadros por segundo. Uma taxa mais elevada, em torno de 60 quadros por segundo, pode ser obtida pela redução da resolução para QVGA (320 x 240 pixels).

A profundidade é codificada através de valores inteiros de 16 bits que representam a profundidade em milímetros. De acordo com a especificação do fabricante, os valores de profundidade variam de 0,8 m a 3,5 m. Uma discussão detalhada da precisão dos dados e

Figura 14 – Asus Xtion PRO LIVE.

Fonte: ASUS (2017)

da calibração das câmeras PrimeSense RGB-D, em particular o Microsoft Kinect, é dado em Khoshelham e Elberink (2012).

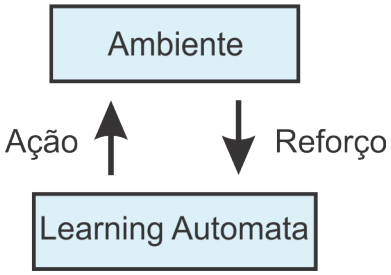
Comparado com o Microsoft Kinect, o Asus Xtion PRO LIVE tem as seguintes vantagens.

1. As imagens RGB e de profundidade são alinhadas em um mesmo sistema de referência;
2. Esta calibração é realizada por uma rotina de alinhamento feita por hardware;
3. O dispositivo da Asus possui menor peso, em torno de 150 gramas, enquanto o Microsoft Kinect pesa aproximadamente, 440 gramas;
4. Sua alimentação é realizada via USB.

2.3 MÉTODOS DE APRENDIZAGEM POR REFORÇO

O *Learning Automata* (LA) é uma abordagem de aprendizagem de máquina muito útil em diversas situações, envolvendo aprendizagem a partir da exploração (HASHEM, 2007). O método seleciona a ação baseada em experiências passadas, aprendidas através da iteração com o ambiente. Em outras palavras, o *Learning Automata* deve ser capaz de coletar e processar as informações adquirida do ambiente, e podendo mudar sua estrutura e parâmetros ao decorrer do tempo para conseguir atingir o objetivo requerido ou o desempenho desejado. A Figura 15 mostra a iteração do ambiente com o *Learning Automata*.

Figura 15 – Interação entre o ambiente e o *Learning Automata*.



O *Learning Automata* teve impacto significativo em várias áreas como em sistemas de controles (HOWELL, 2000), classificação de padrões (ESMAEILPOUR; NADERIFAR; SHUKUR, 2014), controle de sistemas de potência (MISRA *et al.*, 2013), fusão de sensores (LEE, 2016) e etc.

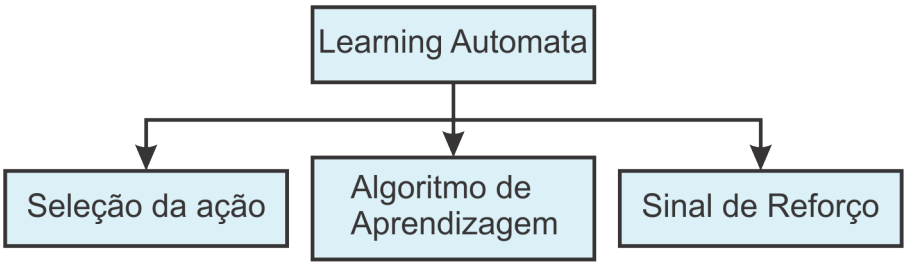
Percebe-se que a aprendizagem concentra-se em três aspectos: Selecionar as ações, aplicar o algoritmo de atualização da probabilidade e receber o sinal de reforço (Figura 16). O algoritmo *Learning Automata* pode ser descrito pelos seguintes procedimentos:

1. Durante a interação com o ambiente, o autômato seleciona randomicamente uma ação do seu conjunto de ações possíveis e que podem ser usadas no estado atual. Essa escolha é baseada em uma distribuição probabilística. As probabilidades de todas as ações são inicialmente iguais;

2. O ambiente então gera uma resposta a ação do autômato, chamado de sinal de reforço. De acordo com a resposta, o autômato atualiza sua distribuição de probabilidade. O algoritmo usado para atualizar a probabilidade das ações é chamado de esquema de atualização do reforço;

3. Uma nova ação é selecionada de acordo com a nova distribuição de probabilidade do autômato. A partir dessa nova ação, uma resposta é gerada pelo ambiente e o procedimento é repetido.

Figura 16 – Composição do *Learning Automata*.



Neste trabalho, duas diferentes abordagens do *Learning Automata* são empregadas, o *Finite Action-Set Learning Automata* (FALA) e *Parameterized Learnig Automata* (PLA).

2.3.1 Algoritmo FALA

A classe dos algoritmos FALA utiliza autômatos para otimizar sua política baseada em um número finito de ações $A = \{a_1, a_2, ..., a_k\}$, onde k é um inteiro e representa o número total de ações. Cada etapa da otimização é chamada de época ou iteração e é dividida em duas etapas: a seleção da ação e a atualização da política de ações.

Inicialmente, em uma época chamada de t, o autômato seleciona randomicamente uma ação $a(t) \in A$ de acordo com uma distribuição de probabilidade $p(t)$. Baseada na ação $a(t)$, o ambiente responde com um sinal de reforço $b(t) \in B$, chamada de recompensa. Assim, o autômato usa esse sinal de recompensa $b(t)$ para atualizar $p(t)$, alterando suas probabilidades e resultando em um novo o valor $p(t+1)$.

Dependendo do conjunto B, diferentes modos de reforço podem ser assumidos. Ambiente de modelo P produz um retorno binário usando $B = \{0,1\}$. No modelo Q o ambiente pode receber um retorno de finitos números entre os valores $B = [0,1]$. No modelo S os valores são números contínuos entre os valores $B = [0,1]$.

Formalmente, o *Learning Automata* é descrito pela quádrupla $\{A, B, \overrightarrow{p}, U\}$, onde:

- A é o conjunto de ações $\{a_1, a_2, ..., a_k\}$ do automato;
- B é o conjunto de reforço do autômato;
- $\overrightarrow{p}$ é o vetor de probabilidades das ações;
- U é o esquema de aprendizagem usado para atualizar o vetor $\overrightarrow{p}$.

Para a atualização da probabilidade, alguns esquemas foram propostos sendo que o esquema do *Reward Penalty* são, provavelmente, os mais usados. A ideia básica por detrás dessa atualização é aumentar a probabilidade de ações que consigam boas recompensas do ambiente, enquanto ações que não possuem boas recompensas tendem a ter um decréscimo em sua probabilidade. Um modo geral do esquema da família dos *Reward Penalty* é dado a seguir (THATHACHAR; SASTRY, 2004):

Se $a_i = a(t)$ tem-se:

$$p_i(t+1) = p_i(t) + \lambda_1 b(t)(1 - p_i(t)) - \lambda_2(1 - b(t))p_i(t) \tag{2.1}$$

Senão:

$$p_i(t+1) = p_i(t) - \lambda_1 b(t) p_i(t) + \lambda_2(1 - b(t))[(k-1)^{-1} - p_i(t)] \tag{2.2}$$

onde, i é uma ação do conjunto de ações, $a(t)$ é a ação selecionada na iteração t e λ_1 e λ_2 são os parâmetros de taxa de aprendizado para recompensa e penalidade respectivamente que são valores em $[0,1]$. Dependendo do λ_1 e do λ_2, o esquema de atualização pode assumir três forma distintas:

1. *Linear Reward-Penalty*(L_{R-P}): Neste esquema as probabilidades das ações são atualizadas em cada etapa. Quando uma recompensa é dada a partir do ambiente, a probabilidade da ação atual é aumentada, enquanto as outras probabilidades são diminuídas. Quando uma penalidade é dada a partir do ambiente a probabilidade da ação atual é diminuída enquanto as outras probabilidades são aumentadas. λ_1 e λ_2 decidem a intensidade da aprendizagem. Elas devem ter o mesmo valor $\lambda_1 = \lambda_2$, mas é possível usar valores diferentes para favorecer a recompensa sobre a penalidade ou vice-versa;
2. *Linear Reward-Inaction*(L_{R-I}) Semelhante ao *Linear Reward-Penalty* mas sem atualizar a probabilidade das ações que sofreram penalidades $\lambda_2 = 0$;

3. *Linear Reward-ε-Penalty*($L_{R\varepsilon P}$) Semelhante ao *Linear Reward-Penalty* mas com λ_2 muito menor comparado a λ_1 ($\lambda_1 >> \lambda_2$), deixando a propriedade do esquema no meio termo entre o L_{R-P} e o L_{R-I};

4. *Linear Inaction Penalty* (L_{I-P}) Este esquema é o oposto do *Linear Reward Inaction*. As probabilidades de ação são atualizadas somente quando uma penalidade é dada do ambiente e as recompensas são ignoradas $\lambda_1 = 0$.

   Para um aprendizado usando o *Linear Reward-Inaction* as Eqs. 2.3 e 2.4 são rescritas para as seguintes equações:

   Se $a_i = a(t)$ tem-se:

$$p_i(t+1) = p_i(t) + \lambda_1 b(t)(1 - p_i(t)) \tag{2.3}$$

   Senão:

$$p_i(t+1) = p_i(t) - \lambda_1 b(t) p_i(t) \tag{2.4}$$

2.3.2 Algoritmo PLA

O algoritmo *Parameterised Learning Automata* (PLA) foi introduzido em Thathachar e Phansalkar (1995). Este método não atualiza diretamente a probabilidade das ações, em vez disso, ele atualiza um vetor u, com um parâmetro u_i correspondente a cada ação a_i. A partir desse parâmetro o vetor de probabilidades pode ser calculado usando alguma função geradora de probabilidade. A função $g(a,u)$ gera a probabilidade da ação selecionada a_i. Uma função comum utilizada para a geração das probabilidades da ação é a função Boltzmann:

$$g(u,i) = \frac{e^{u_i}}{\sum_{j=1}^{k} e^{u_j}} \tag{2.5}$$

O esquema de atualização dos parâmetros do PLA é feito da seguinte forma (VRANCX; VERBEECK; NOWÉ, 2008):

$$u_i(t+1) = u_i(t) + \lambda_1 b(t) \frac{\delta ln(g)}{\delta u_i}(u(t), a(t)) + \lambda_1 h'(u_i(t)) + \sqrt{\lambda_1} s_i(t) \tag{2.6}$$

onde

$$h(x) = \begin{cases} -K(x-L)^{2n} & x \geq L \\ 0 \quad |x| \leq L \\ -K(x+L)^{2n} & x \leq -L \end{cases} \tag{2.7}$$

e $h'(x)$ é a derivada de $h(x)$ e mantêm os valores do vetor de estado em um limite de $|u| \leq L$. O valor λ_1 é a taxa de aprendizagem, as constantes *L, K e n* são todas positivas, sendo L e K números reais e n um número inteiro. O último termo s_i é uma sequência de variáveis aleatórias i.i.d. (independentes e identicamente distribuídas) com media 0 e variância igual a σ^2 que adiciona um ruído para o algoritmo não ficar preso em um mínimo local.

Usando a definição da Eq. 2.5 no gradiente da Eq. 2.6 (HERIK *et al.*, 2007) tem-se:

$$\frac{\delta \ln g}{\delta u_i}(u(t), a(t)) = \frac{\delta}{\delta u_i} \ln \left(\frac{e^{u_{a(t)}(t)}}{\sum_{j=1}^{k} e^{u_j(t)}} \right)$$

$$= \frac{\delta}{\delta u_i} \ln \left(u_{a(t)}(t) - \ln \left(\sum_{j=1}^{k} e^{u_j(t)} \right) \right)$$

$$= \frac{\delta u_{a(t)}}{\delta u_i} - \frac{e^{u_i(t)}}{\sum_{j=1}^{k} e^{u_j(t)}}$$

$$= \frac{\delta u_{a(t)}}{\delta u_i} - p_i(t)$$

$$\frac{\delta \ln g}{\delta u_i}(u(t), a(t)) = \begin{cases} 1 - p_i(t) & se \quad i = a(t) \\ -p_i(t) & nos \quad outros \quad casos \end{cases} \tag{2.8}$$

Substituindo a Equação 2.8 na Eq. 2.6 tem-se:

Se $i = a(t)$:

$$u_i(t+1) = u_i(t) + \lambda_1 b(t)(1 - p_i(t)) + \lambda_1 h'(u_i(t)) + \sqrt{\lambda_1} s_i(t) \tag{2.9}$$

Senão:

$$u_i(t+1) = u_i(t) - \lambda_1 b(t) p_i(t) + \lambda_1 h'(u_i(t)) + \sqrt{\lambda_1} s_i(t) \tag{2.10}$$

Que se assemelha muito ao esquema L_{R-I}.

3 *HARDWARE* E *SOFTWARE*

Este capítulo tem a finalidade de descrever os componentes usados na criação da plataforma robótica para a construção das estruturas, assim como uma breve análise da cinemática do robô. Além disso, são descritas as ferramentas computacionais usadas para a operação do robô real e para a criação do ambiente de simulação.

3.1 PLATAFORMA ROBÓTICA REAL

O robô móvel é composto por rodas omnidirecionais do tipo mecanum (também conhecidas como suecas). Além disso, o robô possui um sensor de visão e uma unidade de processamento embarcada.

3.1.1 Chassi do robô

Para a montagem do robô foi usado um chassi de alumínio, no qual foram acoplados os motores padrão NEMA 17. O chassi também comporta as baterias, a eletrônica de alimentação, o modem *wireless*, os *drivers* dos motores e a unidade de processamento de baixo nível (Arduino Mega). O chassi possui espaçadores para o encaixe de uma base de acrílico. Nessa base de acrílico são fixados os servos motores para a movimentação do braço, a unidade de processamento de alto nível (Raspberry Pi 2) e o sensor de visão (Asus Xtion Pro LIVE).

Os braços foram fabricados a partir de chapas de alumínio com furação para encaixar os servos motores. O terminal do braço possui um furo central para o encaixe do eletroímã. Esse eletroímã é alimentado com uma tensão de 12 V e controlado usando um circuito de potência composto por transistores MOSFET. Esse circuito permite acionar o eletroímã para agarrar a peça quando for necessário. A Figura 17 mostra o robô real construído para realizar a montagem das estruturas.

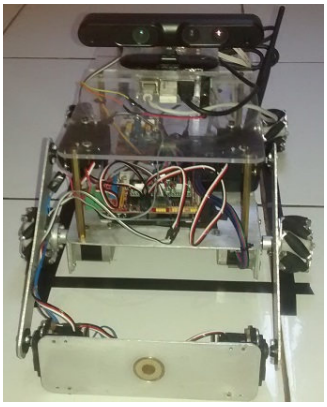
3.1.2 Cinemática

O movimento omnidirecional pode ser alcançado através de quatro rodas mecanum, montadas na base retangular (Figura 18). A formulação das equações cinemáticas de um robô com rodas mecanum são apresentadas em Qioa, Taheri e Ghaeminezhad (2015).

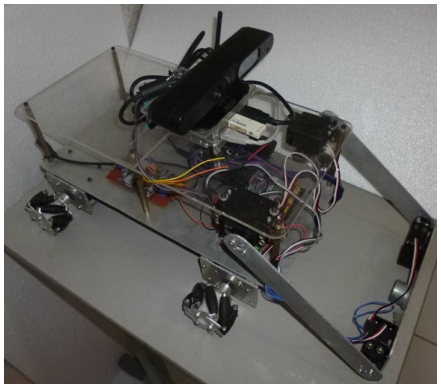
As equações de cinemática inversa e direta são apresentadas a seguir. A equação matricial para encontrar os valores das velocidades de cada roda é dada na Eq. 3.1.

$$\begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{bmatrix} = \frac{1}{r} \begin{bmatrix} 1 & -1 & -(l_x + l_y) \\ 1 & 1 & (l_x + l_y) \\ 1 & 1 & -(l_x + l_y) \\ 1 & -1 & (l_x + l_y) \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ w_z \end{bmatrix} \tag{3.1}$$

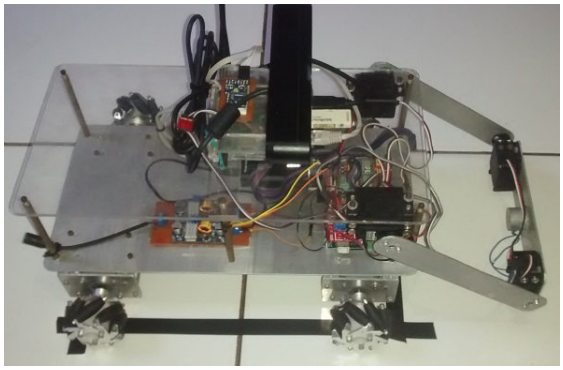
Figura 17 – Robô real com rodas omnidirecionais construído.



(a)



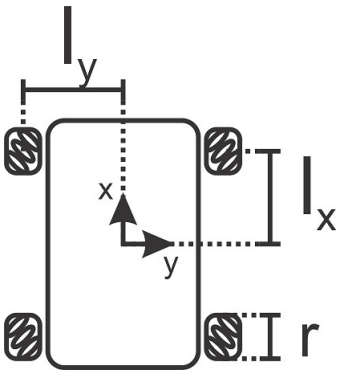
(b)



(c)

Figura 18 – (a) Roda mecanum com seus roletes em 45°. (b) Ilustração da orientação das rodas no robô e variáveis de dimensão

(a)



(b)

Reescrevendo a Eq. 3.1, tem-se:

$$\begin{cases} w_1 = \frac{1}{r}\left(v_x - v_y - (l_x + l_y)\, w_z\right) \\ w_2 = \frac{1}{r}\left(v_x + v_y + (l_x + l_y)\, w_z\right) \\ w_3 = \frac{1}{r}\left(v_x + v_y - (l_x + l_y)\, w_z\right) \\ w_4 = \frac{1}{r}\left(v_x - v_y + (l_x + l_y)\, w_z\right) \end{cases} \tag{3.2}$$

A cinemática inversa (Eq. 3.3) é descrita pela seguinte equação matricial:

$$\begin{bmatrix} v_x \\ v_y \\ w_z \end{bmatrix} = \frac{r}{4} \begin{bmatrix} 1 & 1 & 1 & 1 \\ -1 & 1 & 1 & -1 \\ -\frac{1}{(l_x+l_y)} & \frac{1}{(l_x+l_y)} & -\frac{1}{(l_x+l_y)} & \frac{1}{(l_x+l_y)} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{bmatrix} \tag{3.3}$$

Reescrevendo a Eq. 3.3, pode-se obter um sistema de equações que define a velocidade nas direções X_r e Y_r, além da velocidade angular em torno do eixo Z_r. Note que X_r, Y_r e Z_r são definidos como o sistema coordenadas do corpo (robô).

$$\begin{cases} v_x(t) = (w_1 + w_2 + w_3 + w_4)\frac{r}{4} \\ v_y(t) = (-w_1 + w_2 + w_3 - w_4)\frac{r}{4} \\ w_z(t) = (-w_1 + w_2 - w_3 + w_4)\frac{r}{4(l_x+l_y)} \end{cases} \tag{3.4}$$

onde w_1, w_2, w_3, w_4 são as velocidades de cada uma das rodas, dadas em rad/s. Os valores l_x e l_y são os valores da distância do centro do robô até a roda nas direções X_r e Y_r, respectivamente. O valor r é o valor do raio da roda.

Para controlar o braço é importante saber qual o valor dos ângulos das juntas quando seu terminal está em uma determinada posição. Devido o terminal (θ_{j2}) está sempre em um ângulo de 90 graus com o solo, deve-se determinar somente um ângulo(θ_{j1}). Note que os eixos X_e, Y_e e Z_e são definidos como o sistema de coordenadas do eletroímã.

O ângulo do braço é determinado usando a relação trigonométrica dada através da Eq. 3.5.

$$\theta_{j1} = \arcsin \frac{Z_e - L_{z1} + L_{z2}}{L_a} \tag{3.5}$$

onde θ_{j1} é o ângulo do braço correspondendo a junta 1, Z_e é a posição, onde o sistema do eletroímã está em relação ao eixo Z_r, L_{z1} é a altura da junta 1 em relação ao eixo Z_r, L_{z2} é a altura da junta 2 em relação ao eixo Z_r e L_a é o comprimento do braço do robô.

Com um determinado valor de posição do eletroímã $(x_e, y_e, z_e, \theta_e)$, as seguintes equações são usadas para determinar a posição do centro do robô $(x_r, y_r, z_r, \theta_r)$ e o valor do ângulo do braço θ_{j1}.

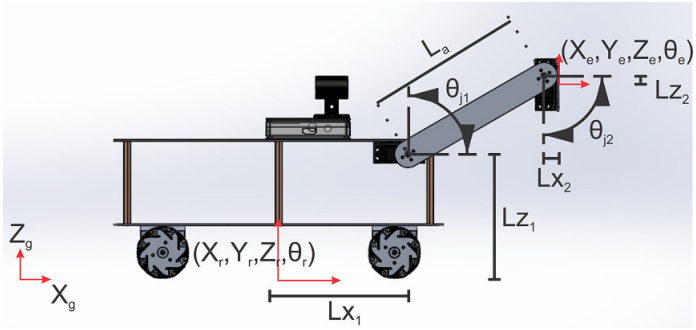
$$x_r = x_e - (L_a \cos\theta_{j1} + (L_{x1} + L_{x2})\cos\theta_g) \tag{3.6}$$

$$y_r = y_e - (L_a \cos\theta_{j1} + (L_{x1} + L_{x2})\sin\theta_g) \tag{3.7}$$

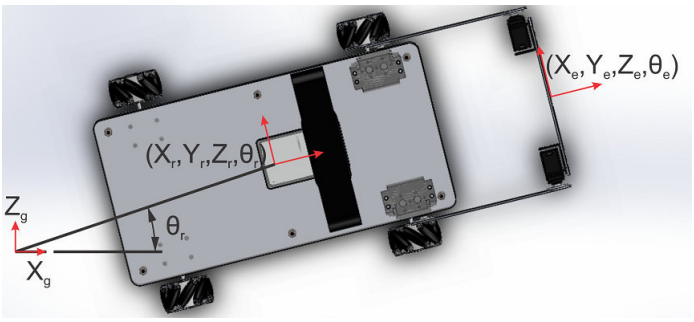
$$\theta_r = \theta_e \tag{3.8}$$

onde x_r e y_r é a posição do centro do robô em relação a coordenada global fixa no ambiente, x_e e y_e são as posições do eletroímã em relação ao sistema de coordenadas global fixa no ambiente., L_{x1} é a distância do centro do robô até a junta 1 ao longo do eixo X_r e L_{x2} é a distância da junta 2 até o eletroímã ao longo do eixo X_r. θ_r e θ_g é o ângulo do robô em relação a coordenada fixa no mapa. A Figura 19 auxilia no entendimento das variáveis usadas. Sendo, X_g, Y_g, Z_g é o sistema de coordenadas global fixo no ambiente.

Figura 19 – Variáveis usadas para encontrar a posição e a orientação do robô a partir de um valor dado para a posição e orientação da terminal



(a)



(b)

3.1.3 Motores e *drivers*

Cada roda do robô é acoplada em um motor de passo padrão NEMA 17. Esse tipo de motor possui a facilidade de precisar de um circuito de acionamento simples. Além disso não são necessários elementos externos para saber o quanto que cada motor rotacionou.

Para o acionamento dos motores foi utilizado o Arduino Mega em conjunto com um *shield*[1] para motores de passo chamado de RAMPS 1.4. O Arduino Mega, via comunicação

[1] Shields são placas que podem ser conectadas na parte superior do PCB Arduino Mega ampliando a sua capacidade.

serial, recebe as informações de velocidade para ser aplicada em cada roda. A velocidade é obtida através da Eq. 3.9:

$$\begin{cases} w_1(passos/s) = w_1(rad/s) \cdot \frac{m_p \frac{360}{\rho}}{2\pi} \\ w_2(passos/s) = w_2(rad/s) \cdot \frac{m_p \frac{360}{\rho}}{2\pi} \\ w_3(passos/s) = w_3(rad/s) \cdot \frac{m_p \frac{360}{\rho}}{2\pi} \\ w_4(passos/s) = w_4(rad/s) \cdot \frac{m_p \frac{360}{\rho}}{2\pi} \end{cases} \tag{3.9}$$

onde m_p é o valor em micropassos configurados no *driver* a4988 e ρ é o valor da resolução do motor de passo (esse parâmetro pode assumir os valores de 0.9º ou 1.8º).

A partir da velocidade requerida por cada roda (Eq. 3.9) é calculado a frequência dos pulsos. O passo no motor é realizado na borda de subida de cada pulso. A Eq. 3.10 descreve como a frequência de pulsos é estimada para cada motor.

$$\begin{cases} f_{m1} = 2 \cdot w_1(passos/s) \\ f_{m2} = 2 \cdot w_2(passos/s) \\ f_{m3} = 2 \cdot w_3(passos/s) \\ f_{m4} = 2 \cdot w_4(passos/s) \end{cases} \tag{3.10}$$

onde $f_{m1}, f_{m2}, f_{m3}, f_{m4}$ são as frequências de pulso aplicadas pelo *driver* a4988.

3.1.4 Alimentação

A alimentação do robô é obtida através do uso de duas baterias LiPo (Polímero de Lítio) em paralelo. Cada bateria possui a capacidade de 2600 mAh e uma voltagem de 11,1 V. A autonomia em funcionamento normal no robô é de aproximadamente 1 hora.

O robô possui três conversores *step-down (buck)*, sendo que o primeiro é ajustado para uma saída de 12 V e os outros dois são ajustados para 5 V. A linha de 12 V é usada para a alimentação dos motores de passo e do eletroímã. As linhas de 5 V são usadas para a alimentação dos servos motores e do modem *wireless* e demais dispositivos. Note que uma dessas linhas é dedicada somente para a alimentação do Raspberry Pi 2.

3.1.5 Câmera RGBD

O sensor de visão usado foi o Asus Xtion PRO LIVE devido ao seu tamanho e peso reduzido em relação ao Kinect. A sua alimentação é feita a partir do USB e seu consumo é inferior a 2,5W. O Asus Xtion PRO LIVE foi posicionado em uma base de acrílico para não sofrer interferência visual dos braços.

3.1.6 Computador embarcado e estação de solo

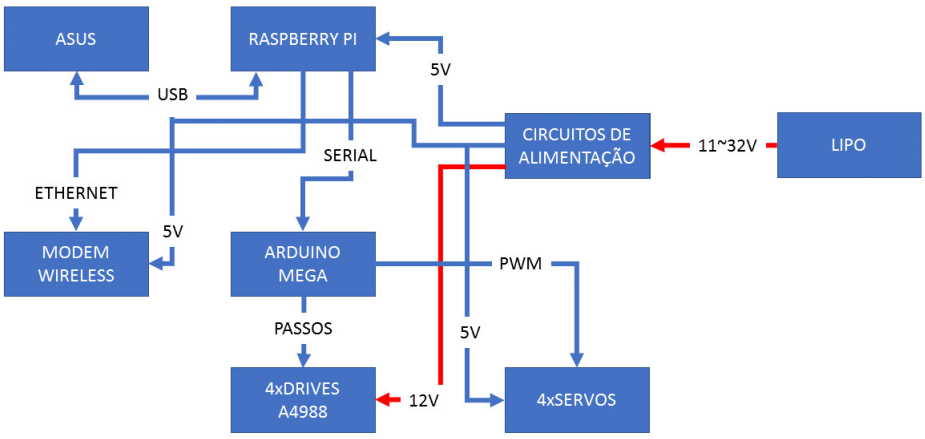
O robô foi equipado com um pequeno computador de bordo (Raspberry Pi 2). A escolha do Raspberry Pi 2 deve-se ao seu elevado poder computacional (que se baseia no processador ARM Cortex M4), ao baixo custo, ao tamanho reduzido e a flexibilidade para a instalação de diferentes sistemas operacionais.

No Raspberry Pi foi instalado uma versão de Ubuntu (16.01LTS) desenvolvido para o Raspberry 2 e 3. Em seguida foi feita a instalação do ROS versão Kinetic. O Raspberry Pi está conectado através de cabo de rede com um modem *wireless* para se comunicar com os outros dispositivos.

Devido ao uso de uma técnica de Visual SLAM para o mapeamento, foi preferível passar a parte de processamento de imagens para uma estação de solo. Em razão disso foi usado um notebook com processador Intel i5 de 1.7 GHz com 6 GB de memória RAM. Neste computador foi instalado o Ubuntu versão 14.04 LTS e o ROS Indigo.

A Figura 20 mostra conexão elétrica e/ou de dados entre os componentes.

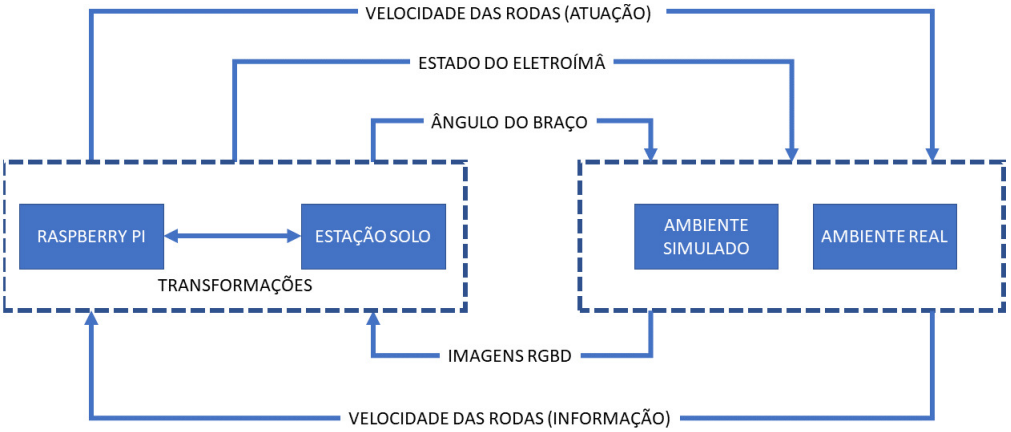
Figura 20 – Conexão entre os componentes usados no robô móvel



3.2 AMBIENTE DE SIMULAÇÃO

O ambiente de simulação foi elaborado para ser o mais fiel ao ambiente real. No ambiente simulado existem propriedades dinâmicas entre os corpos e os efeitos de gravidade. A parte computacional e de controle é feito pelo ROS (abordado na seção 3.3.1). Todos os tópicos criados são acessados pelo robô real e também pelo robô simulado. Para o ROS não existe distinção entre o robô real ou o robô simulado. A Figura 21 mostra as entradas e as saídas dos sistemas real e simulado.

Figura 21 – Simplificação da entrada e da saída do sistema ROS para os ambientes real e simulado.



3.2.1 V-Rep

Esta seção fornece uma visão geral da plataforma de simulação utilizada. O V-Rep é um simulador de robô, desenvolvido e mantido pela Coppelia Robotics (o *download* pode ser realizado em www.coppeliarobotics.com). A versão atual (V3.1.2) permite escolher entre três tipos de *physics engine*, ODE, Bullet e Vortex (somente versão demonstrativa) para simular processos dinâmicos.

O ambiente de simulação é modelado usando o editor de cenas do programa. O V-Rep contém um grande número de modelos predefinidos para diferentes tipos de robôs. O comportamento do simulador é totalmente personalizável e suas funcionalidades podem ser estendida através de *plugins*. Um *plugin* é uma biblioteca de *software* que é escrita em C ++ e interage com o simulador usando a API de programação fornecida.

O V-Rep também permite adicionar *scripts* embutidos para simulações específicas. Esses *scripts* precisam ser escritos na linguagem de programação conhecida como LUA. O V-Rep não é um *software* de código aberto, mas fornece uma licença livre para unidades educacionais e pode, portanto, ser usado para fins de pesquisa.

As simulações no V-Rep são organizadas em cenas. Uma cena representa o mundo simulado que é composto de uma série de elementos, como os corpos tridimensionais, as articulações, os sensores e as câmeras, dentre outros. Esses elementos são combinados em uma estrutura de árvore e são dispostos dentro do ambiente. Essa estrutura de árvore forma a hierarquia de cena. Os elementos dentro dessa hierarquia são chamados de objetos de cena. O V-Rep fornece vários tipos de objetos de cena, mas somente aqueles que são importantes para a implementação são explicados a seguir:

(a) *Shape*: Os *shapes* são caracterizados pelos objetos rígidos, como estruturas dos robôs ou paredes. O V-Rep possui 5 formas primitivas, tais como o cilindro, o cuboide, a esfera, o plano, o disco e as formas complexas como malhas triangulares. Todas essas formas

podem ser combinadas em grupos e, portanto, tratadas como um único objeto. As formas podem ser definidas como estáticas ou não estáticas;

A posição de um objeto estático é fixa em relação ao seu nó pai dentro da hierarquia da cena. Os objetos não estáticos são submetidos à gravidade e cairão se não estiverem conectados a uma junção ou um apoio rígido. Também é necessário distinguir entre formas respondíveis e não respondíveis. As formas respondíveis são manipuladas pelo módulo *physics engine* e criam reações de colisão. Portanto, seus atributos físicos como massa, momento de inércia e configurações de material precisam ser claramente definidos. As formas não respondíveis são visíveis na cena, mas não criam reações de colisão;

(b) Junta: É uma conexão flexível entre duas partes rígidas de um robô. Todas as juntas que foram usadas dentro deste projeto são articulações revolutas, que permitem a rotação em um único eixo. Essas juntas são acionadas por um motor. As configurações do motor incluem limites máximos de torque e de velocidade;

(c) Visão: É realizada a partir de um dispositivo de captura de imagem simulado. Cada sensor de visão captura imagens dependendo de sua localização dentro da cena, do ângulo de visão configurado e da resolução. Os sensores de visão disponíveis são capazes de produzir imagens RGB e de profundidade.

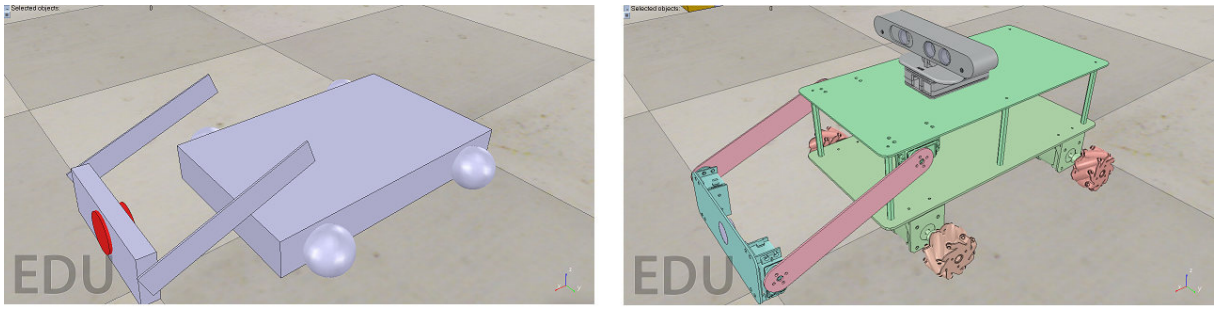
Os modelos de robô são compostos de corpos rígidos chamados de elos, ligados por juntas que permitem acionar as partes flexíveis do robô simulado. Cada *link* é normalmente representado por duas formas diferentes.

A primeira forma refere-se a geometria visual e é geralmente uma forma não respondível. A segunda forma corresponde a geometria de colisão que tem de ser uma aproximação simplificada da geometria visual, composta de grupos de formas primitivas. Essa forma indica a parte respondível do *link* do robô como é visto pelo *physics engine* e precisa, portanto, de uma configuração apropriada dos parâmetros dinâmicos.

A parte respondível é geralmente invisível, mas muito importante, pois aplica comportamento realista ao modelo e permite que ele interaja com outros objetos respondíveis dentro da cena. Sem peças respondíveis, o modelo apenas se movimentaria através de outros corpos, já que apenas formas respondíveis são capazes de produzir reações de colisão.

As dimensões e a parte visual do robô foi criado a partir do software Solidworks e exportados para o V-Rep (Figura 22b). A base inferior e superior do robô assume uma forma primitiva em formato retangular e as rodas usam formas esféricas (Figura 22a). Para cada roda foi adicionada uma junta rotativa, assim como as duas juntas do braço.

Figura 22 – Imagem capturada do V-Rep com a parte respondível do robô em (a) e imagem capturada da parte visual do robô em (b), essa é a imagem visualizada nas simulações.



a b

Na simulação do eletroímã foram usados certos artifícios que não são nativos do V-Rep. Este artifício utiliza um modelo já existente de um sugador a vácuo para funcionar como um eletroímã. Dessa forma ele consegue prender qualquer objeto que se aproxime do terminal do braço quando desejado.

O auto-alinhamento feito por ímãs não é suportado no V-Rep, então por esse motivo, os blocos usados no V-Rep são normais sem nenhuma capacidade de auto-alinhamento.

O V-Rep possui um *plugin* em C++ que consegue fazer a comunicação entre os *scripts* escritos em LUA e os tópicos do ROS. O ROS executado no sistema embarcado no robô possui *scripts* que são usados para publicar mensagens de odometria (*odom*) e mensagens de imagens (RGB e profundidade). Também é feito a leitura dos tópicos responsáveis pelo braço e pelo acionamento do eletroímã.

3.3 FERRAMENTAS COMPUTACIONAIS

3.3.1 ROS

O *Robot Operating System (ROS)* (ROS, 2017) é um *framework* que contém muitas bibliotecas, ferramentas e *drivers* úteis para a criação de aplicações robóticas. O ROS pode atender à demanda de uma ampla gama de robôs de complexidade variada. Além disso, esse software é normalmente instalado na forma de um pacote Debian. Esses pacotes são apenas compatíveis com algumas versões do Ubuntu especificadas na *home page* do ROS[2]. Quando instalado e configurado, o ROS é executado em cima do Linux, tornando-se uma extensão do próprio Linux.

A vantagem do ROS é que ele de uma maneira simples consegue fazer com que aplicações se comuniquem entre si, mesmo quando executadas em máquinas diferentes e/ou usando diferentes linguagem de programação. Cada processo é executado sob o padrão no ROS e representado por um nó. Os nós podem trocar informações enviando ou recebendo mensagens. Essas

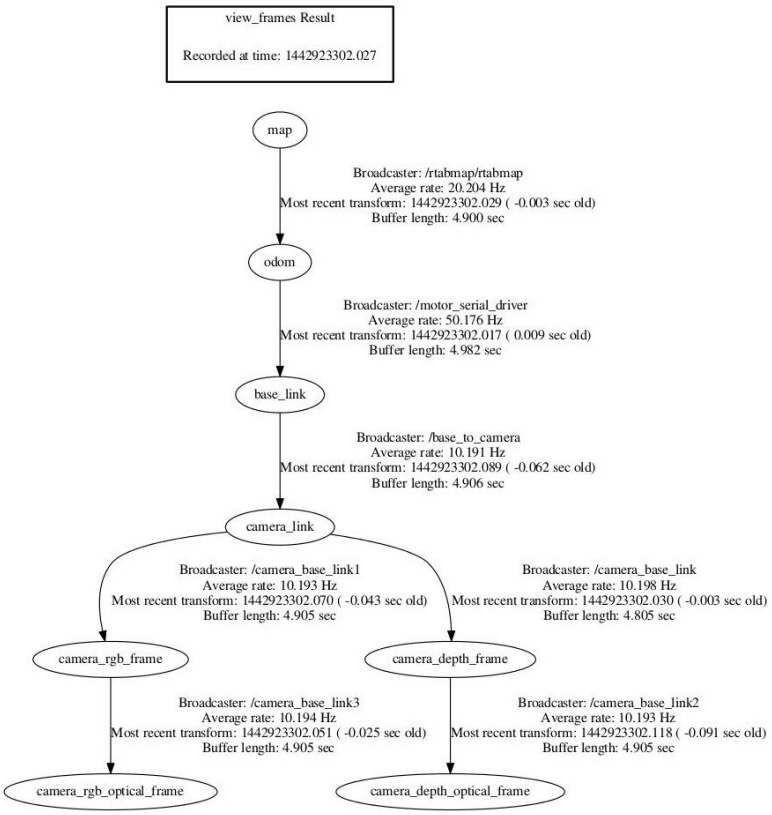
[2] www.ros.org

mensagens podem ser *strings*, números flutuantes, inteiros, vetores e etc. O meio de passagem de tais informações é chamado de tópico. Os tópicos podem ser classificados como *publisher* ou *subscriber*. O *publisher* refere-se ao envio de mensagens e o *subscriber* ao recebimento de mensagens.

Os nós de um sistema ROS podem ser distribuídos em várias máquinas diferentes, mas um deles deve ser definido como nó *Master*. O nó *Master* é responsável por lidar com registros de tópicos e serviços, além de armazenar informações sobre os outros nós do ROS rodando no momento. Outras máquinas podem conectar-se ao *Master* através da rede TCP/IP.

Outra vantagem é a forma como o ROS faz a manipulação de diferentes sistemas de coordenadas. As coordenadas são importantes para saber onde está localizado, por exemplo, o centro do robô, as rodas, os sensores, as juntas e os outros componentes. As transformações entre as coordenadas são publicadas usando o sistema de tópicos. A transformação atual entre duas coordenadas podem ser obtidas lendo as mensagens do tópico *tf*. Um exemplo de árvore de transformação é apresentado na Figura 23.

Figura 23 – Árvore de transformação



A coordenada *map* é uma coordenada global fixa, com o seu eixo Z_g apontado para cima. A pose do robô em relação a coordenada *map* não pode ter escorregamentos ao longo do tempo. A coordenada *map* não é contínua, isso significa que ao longo do tempo pode ter saltos discretos. Em uma configuração normal o algoritmo de localização calcula a pose do robô em relação a

coordenada *map*, eliminando o escorregamento, mas causando os saltos discretos quando novos cálculos são feitos pelo algoritmo de localização. A coordenada *map* é útil para referência global e a longo prazo.

A coordenada *odom* é uma coordenada odometrica que pode ter escorregamentos ao longo do tempo, tornando inútil como referência global. No entanto, a pose do robô em relação a coordenada *odom* é contínua sem saltos discretos. Em uma configuração normal a coordenada *odom* é obtida através de uma fonte de odometria, que podem ser visuais ou obtidas através da odometria das rodas. A coordenada odometrica apresenta uma boa qualidade para referências locais de curto prazo, devido aos escorregamentos por conta dos erros de odometria.

A coordenada *base_link* é a coordenada do corpo que está conectada a estrutura do robô e localizada no centro geométrico do chassi de alumínio. Esse sistema de referência é composto pelo eixo Z_r apontando para cima, o X_r para frente e o Y_r para esquerda, assim quando o robô anda para frente, ele desloca-se ao longo do eixo X_r. Essa convenção é usada devido aos comandos de velocidade usados pelo ROS.

O ROS possui uma biblioteca útil para comunicação com dispositivos via serial. Este pacote é chamado de *serial_ros* e deve ser instalado tanto no Raspbery Pi 2, quanto no Arduino Mega. Com isso, o Arduino Mega consegue criar tópicos e conversar com todo o sistema ROS.

Para a aquisição das imagens do Asus Xtion PRO LIVE, o ROS utiliza o pacote chamado de OPENNI2. Com esse pacote é possível receber as imagens RGB e de profundidade via tópico.

3.3.2 Algoritmo de mapeamento e de localização RTAB-Map

O RTAB-Map (REAL-TIME APPEARANCE-BASED MAPPING) é uma biblioteca RGB-D SLAM baseada em grafo com um detector bayesiano global de fechamento de *loop*. O fechamento de *loop* é a confirmação que o robô retornou a um local previamente visitado. Essa abordagem usa um filtro Bayesiano para avaliar hipóteses de fechamento de *loop*, baseada nas informações de imagens anteriores. A detecção de fechamento de *loop* é essencial para o processo de mapeamento, pois permite que o robô se localize com maior precisão.

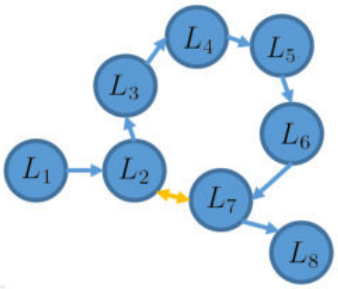
Esta biblioteca usa uma estrutura baseada em grafo que possui nós e arestas para representar o mapa. Os novos locais L_t, representados por nós, são adicionados continuamente à memória de trabalho do sistema ao longo do tempo. Nesse método, as arestas do gráfico são referidas como ligações. Existem dois tipos de arestas: arestas de vizinhança e arestas de fechamento de *loop*.

Arestas de vizinhança são adicionadas entre os nós atuais e os anteriores armazenando as transformações de odometria. Arestas de fechamento de *loop* são adicionadas quando a detecção de fechamento de *loop* é encontrada entre o nó atual e algum nó anterior. Os nós possuem informações das poses de odometria de cada local no mapa, as informações visuais como varreduras a laser, as imagens RGB, as imagens de profundidade e palavras visuais, usadas

para a detecção do fechamento de loop (LABBE; MICHAUD, 2013). As ligações armazenam transformações geométricas estáticas entre nós.

A Figura 24 ilustra o conceito de um grafo criado pelo RTAB-Map ao longo do tempo. Nesse exemplo tem-se o tempo $1 \leq t \leq 8$ com uma hipótese de fechamento de *loop* aceita em t = 7, como mostrado pela seta amarela. L_2 e L_7 são suficientemente semelhantes para serem aceitos como um fechamento de loop.

Figura 24 – Conceito de um grafo criado pelo RTAB-Map ao longo do tempo.



Fonte: Lindrup (2016)

Para a detecção do fechamento de *loop*, é usada a abordagem *Bag-of-Words* para formar um dicionário visual (palavras visuais), usando os algoritmos SURF (*Speeded up Robust Features*) e o SIFT (*Scale Invariant Feature Transform*) (LABBE; MICHAUD, 2014).

Nesta abordagem, um filtro bayesiano é usado para avaliar hipóteses de fechamento de loop sobre todas as imagens anteriores. Quando uma hipótese de fechamento de *loop* atinge algum limiar predefinido H, um fecho de *loop* é detectado. Então as palavras visuais são usadas para calcular a probabilidade requerida pelo filtro. As imagens RGB são combinadas com uma imagem de profundidade, ou seja, para cada ponto 2D na imagem RGB, uma posição 3D correspondente pode ser calculada usando a informação de profundidade dada por uma imagem.

Quando um fechamento de *loop* é detectado, as transformações entre as imagens correspondentes são calculadas por uma aproximação RANSAC (*Random Sample Consensus*) usando as correspondências de palavras visuais 3D. Se for encontrado um mínimo de *inliers*, o fechamento de *loop* é aceito e a aresta com esta transformação entre o nó atual e o nó de hipótese de fechamento de *loop* é adicionado ao gráfico. O RANSAC fornece essencialmente uma resposta, sim ou não, para determinar se o fechamento do *loop* foi concluído.

A abordagem TORO (*Tree-based network optimizer*) é utilizada para a otimização do grafo (Os nós e as transformações são usadas como restrições). Quando os fechamentos de *loop* são encontrados, os erros introduzidos pela odometria são corrigidos e podem então ser propagados para todas as arestas, corrigindo assim o mapa (LABBE; MICHAUD, 2014).

RTAB-Map é distribuído como um pacote ROS e é suportado em ROS Hydro, Indigo, Jade e Kinetic. Ele pode ser usado com uma câmera RGBD ou uma câmera estéreo. O RTAB-

Map também permite usar as informações de odometria como uma suposição inicial. Se não estiver disponível, é possível fazer uma estimativa da posição usando um nó de odometria visual separado (fornecido no pacote RTAB-Map). Porém, essa estimativa pode falhar devido ao baixo número de características detectadas. Usando a odometria das rodas (ou combinação dos dois) é, portanto, mais seguro e robusto.

O robô possui as informações de odometria das rodas que estão disponíveis via tópico, essa odometria é usada como estimativa inicial para o RTAB-Map. Como se sabe a odometria possui erros que ao longo da trajetória vão se somando. Então, o RTAB-Map corrige a transformação da coordenada *odom* em relação a coordenada *map*, para garantir a estimativa da posição correta da coordenada *base_link* em relação a coordenada *map*.

3.3.3 Algoritmo de controle e navegação

O ROS fornece recursos para navegação 2D. Um planejador global utiliza o ponto inicial e final de um procedimento de montagem para gerar uma trajetória segura. O planejador global é composto por um algoritmo A*. O controlador de rastreamento de trajetória fornece o controle a base móvel do robô para fazer sua movimentação no plano global. São disponíveis alguns controladores: O *dwa loca planner* (FOX; BURGARD; THRUN, 1997), o *eband local planner* (QUINLAN; KHATIB, 1993) e o *teb local planner* (ROESMANN *et al.*, 2012). Foi usado o *eband local planner* por ter melhores resultados com a configuração móvel usada.

O *eband local planner* possui capacidade de realizar desvios de obstáculos e criar trajetórias durante o percurso. Mas, neste trabalho, tais recursos foram desativados, pois o planejador global gera uma trajetória livre de obstáculos.

Para realizar a navegação algumas etapas são realizadas. A primeira é a definição do ponto no qual o robô deve ir. O controlador (*eband local planner*) envia os comandos de velocidade para fazer o robô chegar ao ponto desejado, usando a posição do robô em relação a coordenada *map*. A velocidade aplicada ao robô é enviada pelo tópico *cmd_vel*. A conversão da velocidade do robô na coordenada do corpo para as velocidades aplicadas em de cada roda é feita pelo nó *velocidade rodas*. O nó *velocidade rodas* é responsável por escutar esse tópico e transmitir para o Arduino Mega via tópico, as velocidades das rodas individualmente. Essa conversão é feita a partir da Eq. 3.9.

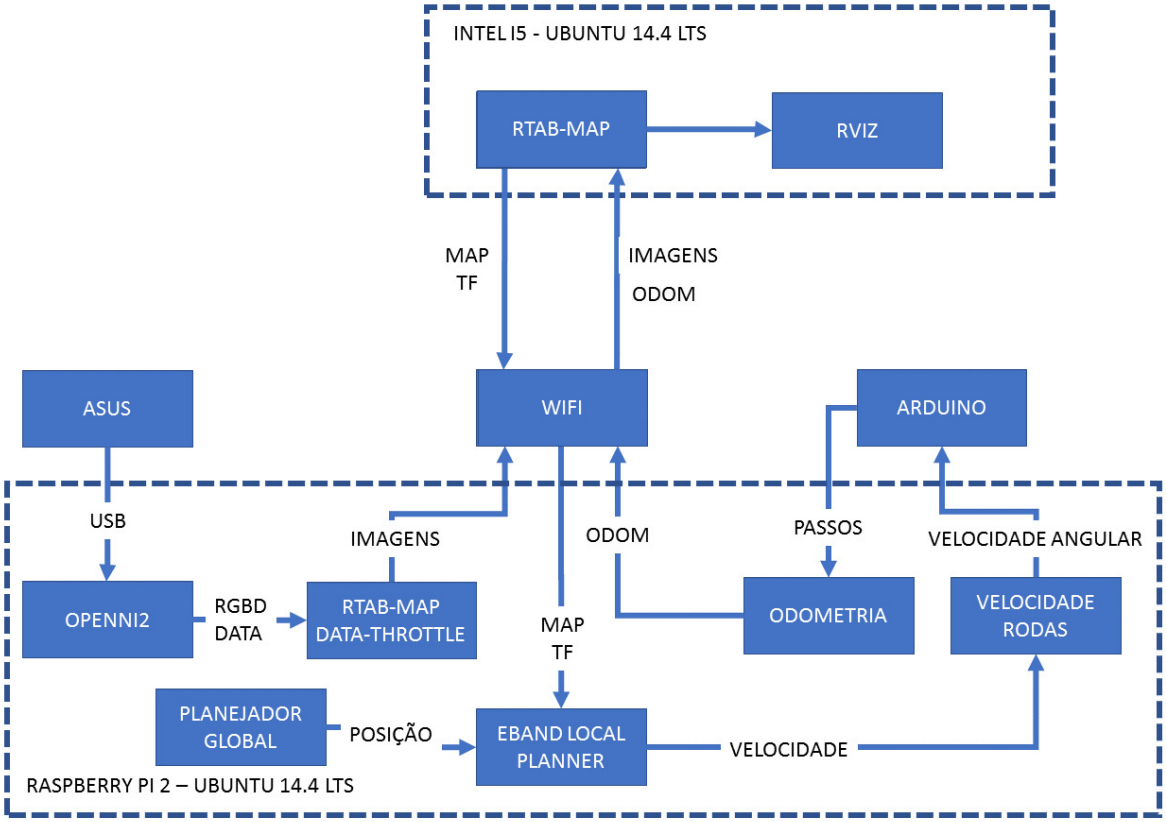
O Arduino Mega, ao receber o valor de velocidade do Raspberry Pi 2, aplica estes valores aos motores de passo. O Arduino Mega envia para o Raspberry o número de passos que foram aplicados no motor de passo em uma frequência de 30 Hz. As velocidades das rodas são calculadas a partir dos números de passos enviados pelo Arduino Mega. O nó odometria converte as velocidades das rodas em velocidades dos eixos do robô usando a Eq. 3.4.

Enquanto o robô se desloca, o Asus Xtion PRO LIVE adquire as imagens do ambiente e, em seguida, essas imagens são enviadas via USB para o Raspberry Pi 2. O nó OPENNI2 gera

os tópicos referente a imagem. Esses tópicos são lidos pelo nó do RTAB-Map que está sendo executado no Raspberry Pi 2. O nó do RTAB-Map é usado para diminuir a resolução e a taxa de atualização das imagens para uma taxa de 5 quadros por segundo. Esses dados são comprimidos e junto com a odometria são enviados para a estação de solo via *wifi* por tópicos.

A estação de solo é usada para executar o RTAB-Map junto com a ferramenta de visualização. A imagem e a odometria são lidos e processados pelo RTAB-Map que gera a posição do robô, em relação a coordenada global *map*. Esses dados são enviados para o Raspberry Pi 2 e lidos pelo controlador *eband local planner*, o controlador recebe sua posição global e pode corrigir os erros de posição. Esse processo é realizado em tempo real até que a posição seja alcançada com um erro mínimo em comparação a posição real. A Figura 25 ilustra as etapas mencionadas.

Figura 25 – Conexão entre os diferentes nós usados no ROS para o funcionamento do robô.



4 CONSTRUÇÃO USANDO UM ROBÔ TERRESTRE

Neste capítulo são abordados os materiais usados na confecção dos blocos de montagem, a classificação quanto as suas dimensões e o local de armazenagem em relação a uma coordenada global. Além disso são apresentados os tipos de estruturas a serem construídas e o ambiente usado para construção no modelo simulado e no modelo real.

O conjunto de blocos forma uma estrutura. No entanto, para a montagem de uma estrutura, um conjunto de condições deve ser atendido. Entre tais condições pode-se destacar a posição dos blocos em relação a um bloco vizinho no mesmo nível ou em um nível superior ou inferior. Assim, este capítulo descreve como deve ser planejado a alocação dos blocos para a construção de estruturas.

O planejamento da tarefa concentra-se na definição do plano de execução para os diferentes tipos de estruturas. Um plano de execução mal planejado pode resultar em uma construção impossível do robô realizar.

4.1 INSUMOS PARA A CONSTRUÇÃO

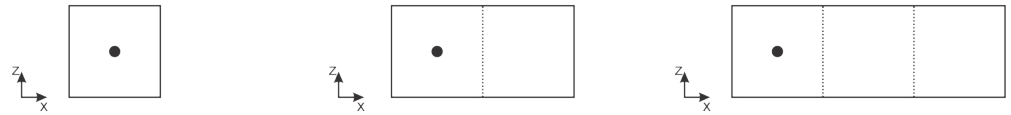
Em um cenário típico de construção de estruturas é possível imaginar que diferentes configurações e tamanhos de estruturas possam ser construídas. Para que isso ocorra, a melhor opção é que esteja disponível no ambiente um conjunto de peças de diferentes tipos, para que diferentes estruturas possam ser construídas.

Neste trabalho, o conjunto de peças usadas para a construção das estruturas tridimensionais são blocos confeccionados a partir de folhas de poliestireno.

Os blocos são classificados em relação ao seu comprimento. O bloco mais simples têm o formato de um cubo de dimensão 6 x 7 x 7 cm e é chamado de bloco de uma unidade (simples). No entanto, o espaço reservado para a montagem do bloco simples na estrutura é de 8 x 8 x 8 cm. Os blocos precisam ser menores devido ao modo como o robô encaixa os blocos nas estruturas. Se não existir um espaço livre entre os blocos, o bloco que está sendo montado pode entrar em contato com os outros blocos montados e, possivelmente, destruirá a estrutura em construção. Para uma melhor margem, o eixo X possui uma redução maior quanto aos eixos Y e Z. A coordenada de origem do bloco está localizada no seu centro geométrico (Figura 26a).

Blocos de dimensões maiores são classificados de acordo com o seu tamanho em relação ao bloco de uma unidade. O bloco de duas unidades é a junção de dois blocos simples, resultando em uma peça de 12 x 7 x 7 cm, considerando a margem de segurança para o encaixe das peças. Então a mesma lógica é usada para blocos de três, de quatro unidades ou maiores. A coordenada de origem dos blocos de tamanho maior continuam na mesma posição dos blocos menores. A Figura 26 ilustra os blocos de diferentes tamanhos, assim como a localização dos pontos de origem de cada bloco.

Figura 26 – (a) Bloco simples; (b) Bloco de duas unidades; (c) Bloco de três unidades.



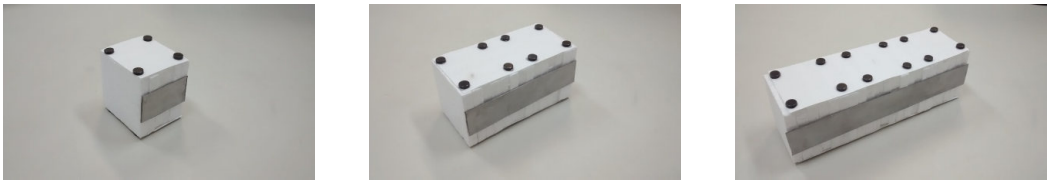
(a) Simples (b) Duas unidades (c) Três unidades

Uma característica que facilita a montagem dos blocos é definida como autoalinhamento. O autoalinhamento é a possibilidade de um bloco superior alinhar-se de formar pré-determinada a partir de um bloco inferior. O alinhamento é realizado através de ímãs circulares, pequenos, montados nas superfícies superiores e inferiores de cada bloco.

As disposições dos ímãs na superfície do bloco formam um retângulo. O alinhamento é possível quando os blocos estão a uma distância mínima para que os ímãs possam realizar o alinhamento e o acoplamento das peças.

Os blocos disponíveis no ambiente são agarrados pelo robô através do eletroímã montado no terminal do braço. No entanto, esses blocos devem ser compostos por uma tira metálica em sua lateral para que sua captura seja possível. A Figura 27 ilustra os blocos de diversos tamanhos, assim como a disposição dos imãs em sua superfície e o local da fixação da tira metálica.

Figura 27 – Todos os blocos com seus imãs e placa metálica posicionada.



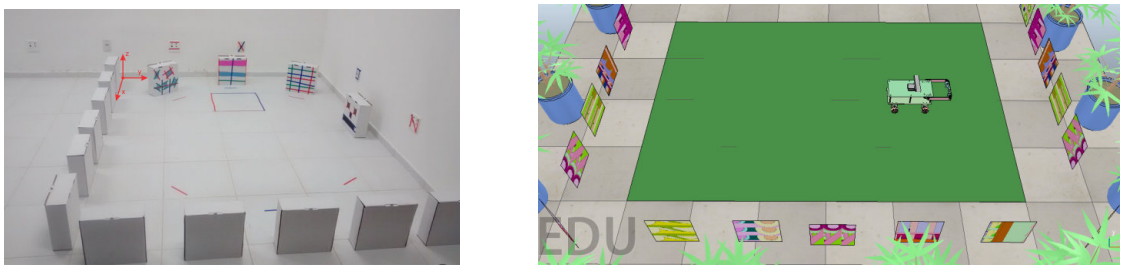
(a) Simples. (b) Duas unidades (c) Três unidades

4.2 AMBIENTE DE MONTAGEM

A Figura 28 mostra a configuração do ambiente proposto para a realização da montagem da estrutura. Os eixos X_g, Y_g e Z_g são usados como sistema de coordenada global. A estimativa de posição e de orientação do robô móvel, assim como a posição e a orientação dos blocos é feita em relação a esses eixos. Na Figura 28a é mostrado o ambiente real e na Figura 28b o ambiente simulado no V-Rep.

O espaço de trabalho útil disponível para a montagem das estruturas é chamado de área de montagem e no ambiente proposto tal área está localizada no centro do ambiente. A área de manipulação é a região onde são disponibilizados os blocos. Estes locais encontram-se nos quatro cantos do ambiente.

Figura 28 – Ilustração do ambiente real e do ambiente simulado.

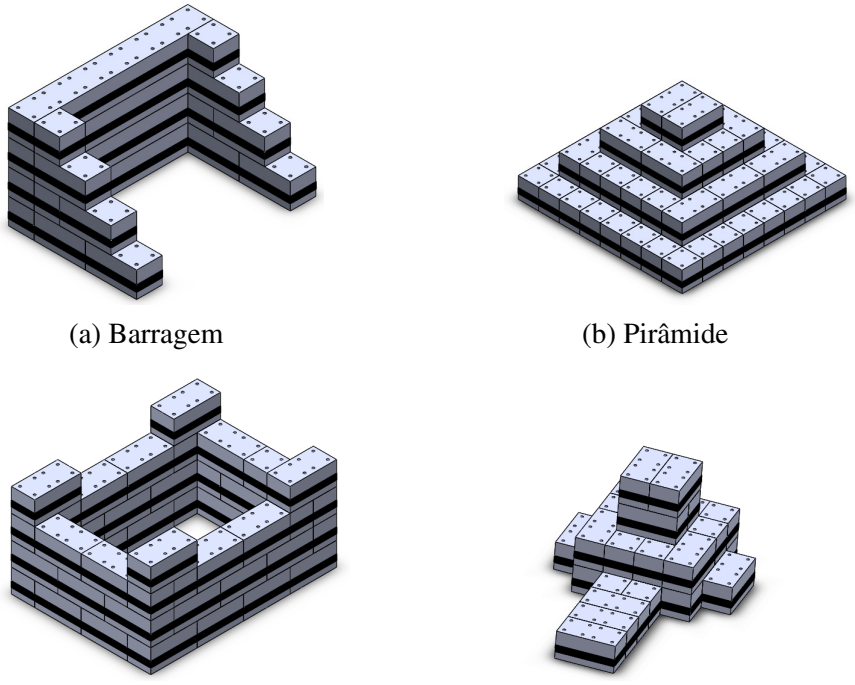


(a) Ambiente real (b) Ambiente simulado

4.3 TIPOS DE ESTRUTURAS

Nesta seção são apresentadas as estruturas que são investigadas neste trabalho. Estas estruturas são semelhantes as estruturas frequentemente utilizadas em diferentes setores da sociedade, como construções arquitetônicas e para fins comerciais e militares. A Figura 47 ilustra algumas das estruturas possíveis que podem ser construídas a partir dos blocos de diferentes tamanhos. Todas elas possuem, 4 níveis de altura e usam blocos no sentido vertical ou horizontal, visto no plano XY.

Figura 29 – Representação de estruturas usando blocos de três, de duas e de uma unidade.



(a) Barragem (b) Pirâmide

(c) Torre (d) Estação Espacial

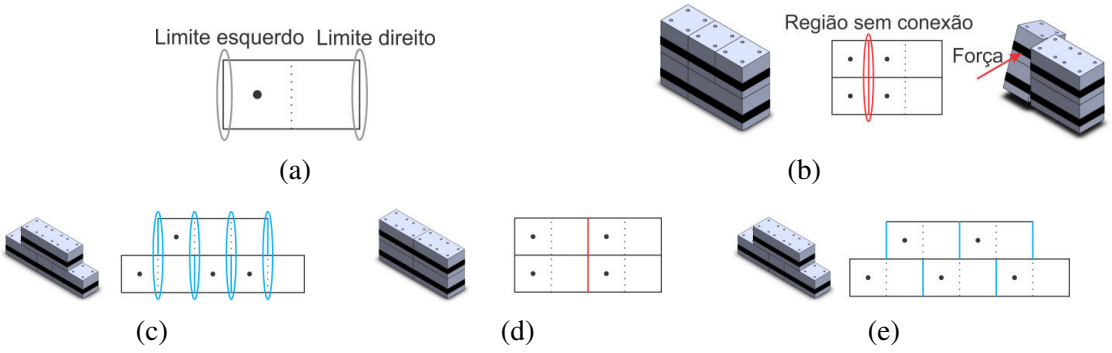
O que pode-se notar na construção, seja na indústria ou na construção civil (alvenaria), é que sempre deve ser garantida a estabilidade das estruturas durante a sua montagem. Em outras palavras, deve-se garantir que, ao se colocar um novo bloco, a parte da estrutura já construída não seja comprometida.

Para garantir essa estabilidade os blocos devem ser posicionados de forma entrelaçada, ou seja, deve-se colocar os blocos de maneira que as camadas superiores e inferiores fiquem defasados entre si, garantindo uma maior área de contato entre os blocos e evitando regiões com falhas de conexão.

A região mais esquerda do bloco é nomeada de limite esquerdo do bloco e a parte mais à direita é nomeada de limite direito do bloco (Figura 30a). As regiões com falhas de conexão ocorrem quando o limite esquerdo ou direito do bloco superior coincide com o limite esquerdo ou direito do bloco montado na camada inferior.

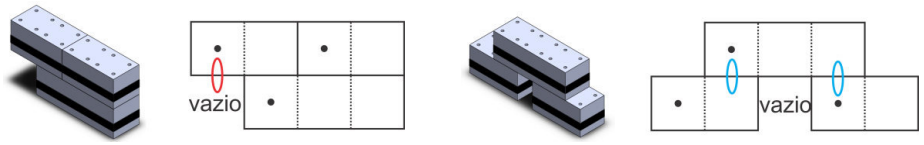
Na Figura 30b são ilustradas áreas marcada em vermelho mostrando uma região com falha de conexão resultando em uma estrutura mais suscetível a danos com aplicações de forças externas. A Figura 30c ilustra uma área marcada de azul mostrando a boa conexão do bloco superior com o inferior. A Figura 30d e a Figura 30e mostram uma estrutura com falhas de conexão e boas conexões respectivamente.

Figura 30 – Representação da conexão dos blocos.



Além de evitar as regiões com falhas de conexão, as estruturas não podem possuir blocos superiores suspensos e apoiado apenas em um dos lados, ou seja, com a área suspensa maior que a área apoiada. As regiões suspensas devem ser suportadas por, no mínimo, duas áreas. A Figura 31 ilustra esses dois casos.

Figura 31 – Região suspensa não permitida em (a) e em (b) uma região suspensa permitida.



(a) Bloco somente com o apoio na área esquerda

(b) Bloco com apoio na área esquerda e direita

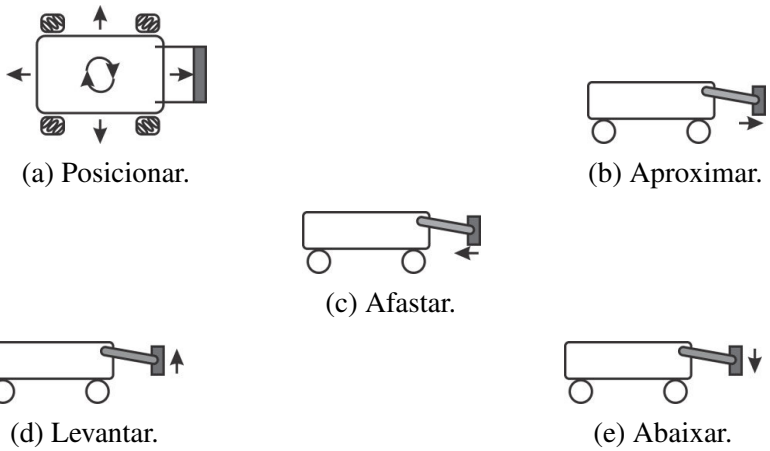
Note que a definição dos tipos de blocos usados para a construção da estrutura, o entrelaçamento entre os diferentes níveis e a verificação das regiões de conexão entre os blocos são estabelecidos pelo algoritmo de planejamento apresentado no Capítulo 5.

4.4 PROCEDIMENTOS DE MANIPULAÇÃO E DE MONTAGEM

O robô é capaz de capturar os blocos em seus locais de armazenamento e montar a estrutura planejada obedecendo o plano de execução. Para isso, o robô usa um conjunto de ações para realizar as tarefas. Ao todo, cinco possíveis ações podem ser realizadas. A primeira ação é definida como Posicionar, como mostra a Figura (Figura 32a), essa ação consiste em posicionar o robô em algum ponto do mapa. Além disso, essa ação possui subrotinas que corresponde ao deslocamento do robô nos eixos X_r, Y_r e torno do eixo Z_r. Essas subrotinas são usadas para mover o robô de um ponto a outro ou mudar a orientação do robô.

A segunda ação é definida como Aproximar (Figura 32b). Essa ação executa o deslocamento do robô no eixo X_r com uma velocidade reduzida de um ponto ao outro. Tal deslocamento é usado para entrar na área de montagem ou de manipulação. A terceira ação é definida como Afastar (Figura 32c). Essa ação executa o deslocamento do robô no eixo X_r com uma velocidade reduzida de um ponto ao outro e é usada para sair da área de montagem ou de manipulação. A próxima ação é definida como Levantar (Figura 32d). Essa ação consiste em levantar o braço até o ângulo desejado. E por fim, a ação definida como Abaixar (Figura 32e), que consiste em abaixar o braço até um ângulo desejado.

Figura 32 – Ações possíveis nas tarefas de manipulação e montagem



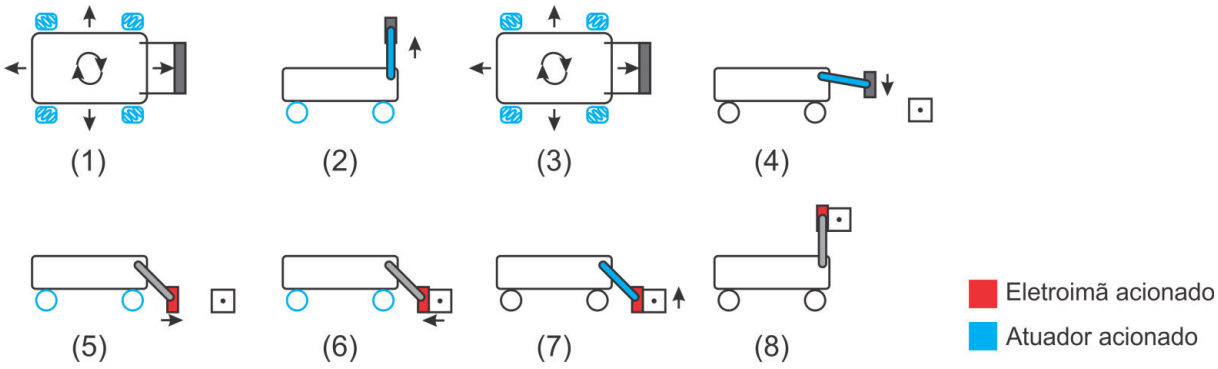
(a) Posicionar. (b) Aproximar. (c) Afastar. (d) Levantar. (e) Abaixar.

A primeira tarefa do robô é fazer a trajetória até o primeiro bloco. Todos os procedimentos realizados pelo robô até o momento que o bloco é capturado denomina-se procedimento de manipulação. O procedimento de manipulação consiste em realizar uma trajetória sem o bloco. Essa trajetória é feita com o eletroímã desligado e com o braço erguido, em um ângulo de 90 graus em relação ao solo. Note que o gasto de energia para manter o braço nessa posição é menor.

Quando o robô chega na área de armazenagem, ele alinha sua garra de forma que a garra magnética e a face posterior do bloco fiquem em paralelo. Com o alinhamento realizado, o braço é abaixado e o robô realiza a aproximação com o eletroímã ligado. Após capturar o bloco, o robô afasta-se da área de manipulação e ergue os braços.

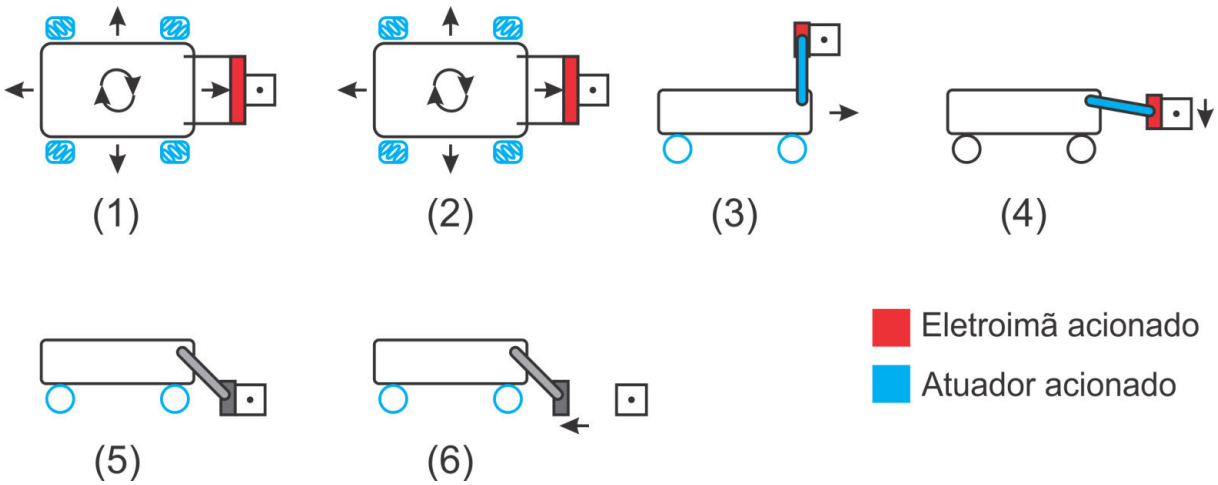
Quando o bloco é capturado e o braço erguido, o procedimento de manipulação chega ao fim. Ao final de cada ação, o robô deve esperar um pequeno intervalo de tempo antes de executar a próxima ação. A Figura 34 mostra o conjunto de ações que compõem o procedimento de manipulação.

Figura 33 – Procedimento de manipulação



Com o fim do procedimento de manipulação inicia-se o procedimento de montagem. O procedimento de montagem consiste em levar o bloco da área de armazenamento até o seu ponto de montagem. A trajetória é feita com o eletroímã ligado e com o braço erguido. Ao chegar próximo a área de montagem, o robô realiza seu alinhamento com o ponto de montagem. Assim que alinhado, o robô aproxima-se do ponto desejado e começa a abaixar o braço, até que o bloco esteja na altura do ponto de montagem. Em seguida, o eletroímã é desligado e o bloco é depositado no local de montagem. Ao finalizar o procedimento de montagem, o robô deve receber outro procedimento de manipulação e assim continuar até que o todo plano de montagem tenha sido executado. A Figura 34 apresenta o conjunto de ações que compõem o procedimento de montagem.

Figura 34 – Procedimento de montagem.



4.5 PLANO DE EXECUÇÃO

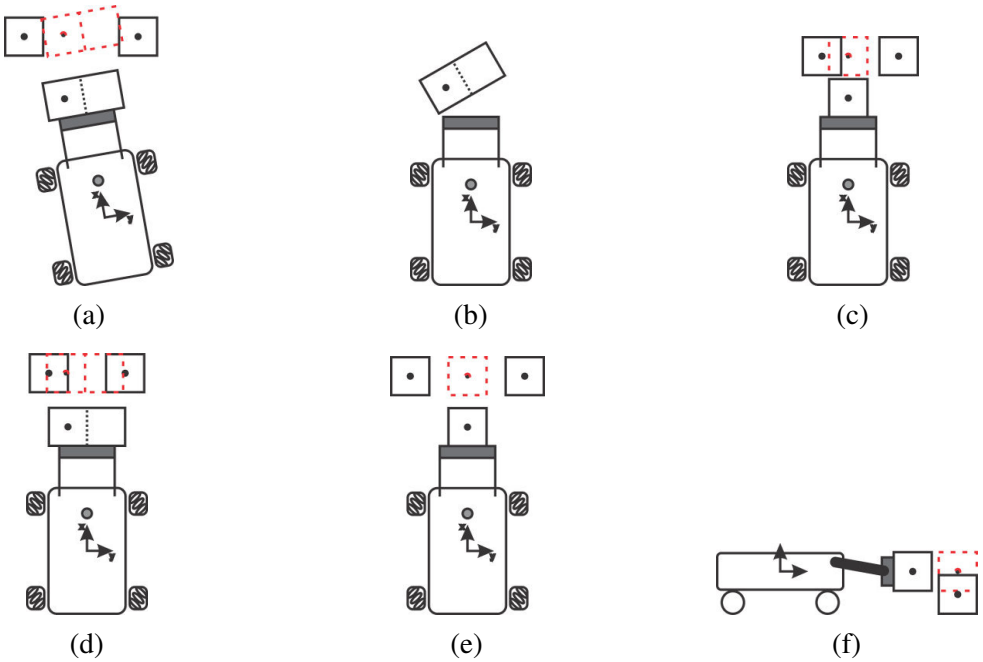
O plano de execução é uma representação não visual da estrutura. O plano de execução é um conjunto de ações informando ao robô como deve ser construída a estrutura. O plano começa com a primeira ação e assim continua até que a última ação tenha sido executada.

A primeira ação a ser executada é a de manipulação. Nessa ação é informado a posição e a orientação (X, Y, Z, θ) do bloco armazenado na área de manipulação. Essa ação gera um procedimento de manipulação. A segunda ação é a de um procedimento de montagem. Nessa ação é informada a posição e a orientação do local onde o bloco é alocado na área de montagem. Essa ação gera uma tarefa de montagem.

Note que nem todo plano de montagem pode de ser realizado com o robô móvel proposto devido as suas limitações geométricas. As Figuras 35 e 36 ilustram alguns casos em que uma montagem pode não ser bem-sucedida.

A Figura 35b ilustra o caso do robô tentando capturar uma peça sem estar com a garra em paralelo com a face posterior do bloco. Essa situação pode ocasionar em uma falha durante a captura do bloco. Na Figura 35a percebe-se que o robô tenta colocar o bloco em um ângulo diferente da orientação dos outros blocos. Se isso ocorrer, o bloco irá ocupar a posição em vermelho pontilhado. Essa posição entra em conflito com os outros blocos já montados. Neste trabalho, as orientações definidas para colocação dos blocos são de 0, 90, 180 e 270 graus.

Figura 35 – Situações que devem ser evitadas durante a montagem de estruturas.



No caso das Figuras 35c e 35f, o robô tenta colocar o bloco em uma posição de montagem diferente da que é possível montar. Nesses casos, a tentativa de alocar o bloco em tal posição

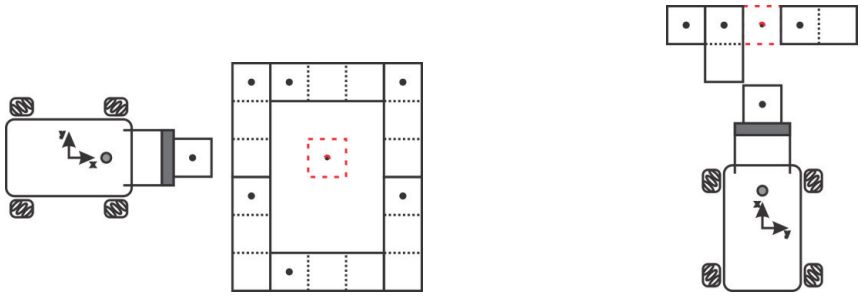
pode gerar a colisão do bloco na garra com os blocos já montados.

As Figuras 35d e 35e são situações em que o bloco usado não tem o tamanho correto para o ponto de montagem. Essas duas situações podem ocorrer se:

1. O bloco for maior que o espaço reservado. Isso pode causar a colisão do bloco agarrado com os blocos já montados (Figura 35d);
2. O bloco for pequeno ou possuir uma folga excessiva. Isso pode causar problemas durante a montagem dos níveis superiores (Figura 35e).

Vão existir casos que uma estrutura não pode ser montada, se não for respeitada a ordem de instalação das peças estabelecida pelo plano de montagem. A Figura 36a mostra duas inconsistências de montagem que devem ser evitadas. No primeiro caso (Figura 36a), o robô recebeu a tarefa de colocar o bloco na área pontilhada de vermelho, mas como a já existem blocos arredor, o procedimento de montagem não pode ser realizado. Esse problema pode ser resolvido mudando a ordem de montagem, ou seja, começando pela colocação dos blocos internos para que, em seguida, sejam colocados os blocos externos.

Figura 36 – Situações em que não é possível posicionar o bloco.



(a) (b)

Outro problema decorrente da ordem da montagem é visto na Figura 36b. Nesse caso o robô não consegue colocar o bloco porque isso resulta no contato de um bloco já posicionado, podendo causar danos a estrutura montada. Esse problema pode ser resolvido mudando a orientação que o robô posiciona o bloco. No caso da Figura 36 , a montagem torna-se valida, se o robô posicionar o bloco seguindo o sentido de cima para baixo.

5 PROCESSO DE APRENDIZAGEM DA TAREFA DE MONTAGEM

Neste capítulo é apresentado uma abordagem que utiliza o aprendizado por reforço para o planejamento da montagem autônoma de estruturas de tridimensionais, especificada pelo usuário. O sistema de montagem autônomo é dividido em três etapas:

1. Definir a estrutura em um modelo digital, no qual será usada com base para a geração do diagrama ótimo de montagem;

2. Gerar um diagrama ótimo de montagem, que consiste em uma lista de posições, orientações e tipos de blocos, respeitando a forma final especificada pelo usuário. Esse diagrama é gerado considerando a minimização da quantidade de blocos usados e obedecendo as restrições quanto ao posicionamento dos blocos;

3. Gerar de um plano ótimo de execução, que é usado pelo robô para realizar a tarefa de montagem da estrutura. Esse plano consiste em uma sequência ótima de procedimentos de manipulação e de montagem dos blocos. Essas sequências devem resultar em uma montagem de menor tempo possível, levando em consideração a geometria do robô e a potência dos atuadores.

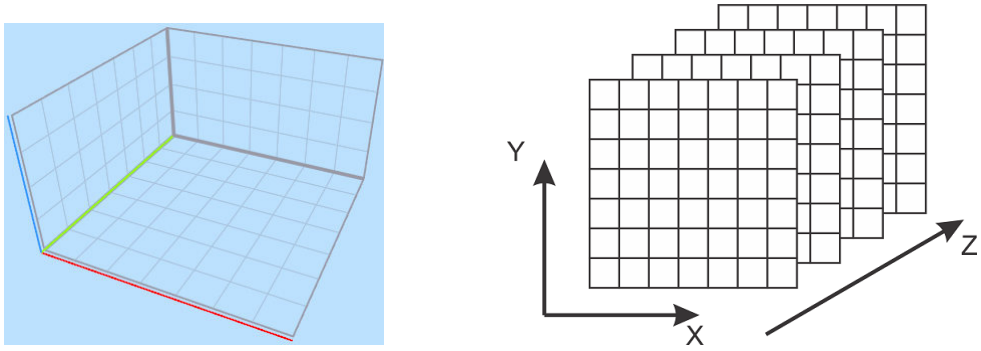
5.1 ENTRADA DO SISTEMA

Primeiramente para resolver o problema de montagem deve-se representar digitalmente a forma final desejada para a estrutura. O modelo digital possui *grids* de tamanhos iguais a dimensão do bloco de uma unidade, o valor da aresta do bloco é definida pela variável φ. A Figura 37a mostra uma área de montagem de dimensões 7 x 7 x 4 representada por *grids*.

Usando a representação digital, as estruturas podem ser construídas usando blocos de uma unidade. Assim, o usuário não precisa se preocupar com os blocos de diferentes tamanhos.

A representação digital pode ser traduzida na forma de uma matriz tridimensional binária chamada de M_{ent} e possui dimensões iguais a $(M_{Ztotal}, M_{Ytotal}, M_{Ztotal})$ (Figura 37b). Na matriz M_{ent}, o valor 0 refere-se a uma célula do *grid* não ocupada por um bloco e o valor 1 representa a célula ocupada. Essa é a forma mais simples de representar uma estrutura, usando apenas os blocos de 1 unidade.

Figura 37 – Área de montagem em forma de *grids* e sua matriz tridimensional.



(a) *Grid* vazio para a construção (b) Matriz tridimensional

Mesmo simplificando a entrada do sistema para uma matriz binária, ainda torna-se bastante difícil definir valores (iguais a 0 ou 1) para cada célula dessa matriz. Principalmente em estruturas muito grandes.

Existem *softwares* livres ou pagos para criações em três dimensões usando *voxels*. Um *voxel* é uma unidade de informação gráfica que define um ponto no espaço tridimensional, assim como o *pixel* (elemento de imagem) define um ponto no espaço bidimensional. Um *voxel* representa um ponto em um espaço tridimensional.

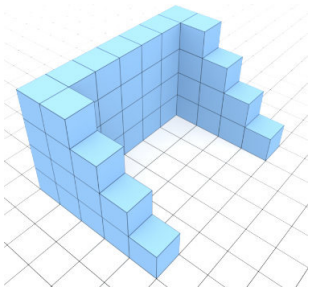
O *software* MagicaVoxel trata o *voxel* como uma unidade gráfica em forma de cubo. Esse programa oferece recursos para adição de blocos, remoção, visualização e tarefas mais complexas. Ele pode trabalhar com uma extensão de arquivo chamada de vox. As informações dos arquivos salvos em vox são compostas por uma lista de células do *grid* que estejam ocupadas por um bloco. Realizando uma simples operação é possível traduzir a lista de células ocupadas para uma matriz M_{ent}.

Para que seja possível montar uma estrutura ela não pode conter blocos suspensos, ou seja um bloco em um nível superior sem que exista um bloco no nível inferior, com exceção dos blocos que formam a base da estrutura.

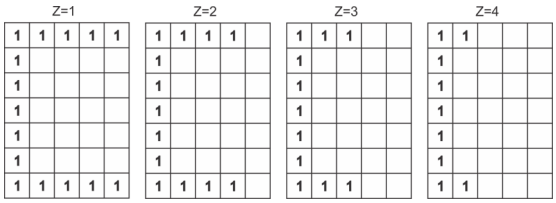
Para usar o MagicalVoxel primeiramente deve-se escolher os limites máximos de construção $(M_{Ztotal}, M_{Ytotal}, M_{Ztotal})$, em seguida usar a ferramenta para adicionar os blocos no ambiente (Figura 38a). No fim da criação digital da estrutura é necessário salvar o arquivo no formato vox.

Um programa escrito em MATLAB, desenvolvido neste trabalho, chamado de *traduzVOX*, é usado para ler os arquivos no formato vox e traduzir em uma matriz binária tridimensional (Figura 38b). Essa tradução é armazenada como variável do ambiente para poder ser usada nas etapas seguintes do processo de montagem da estrutura.

Figura 38 – Representação digital de uma estrutura e sua matriz M_{ent}.



(a) Estrutura criada no MagicalVoxel.



(b) Matriz de entrada da estrutura criada no MagicalVoxel.

5.2 GERAÇÃO DO DIAGRAMA DE MONTAGEM

Para geração do diagrama de montagem foi modelado o sistema usando autômatos. O modelo composto por uma matriz tridimensional de autômatos chamada de M_{aut}, que possui as mesmas dimensões da matriz M_{ent}. A matriz M_{aut} possui autômatos nas células que correspondem ao valor 1 na matriz M_{ent}. A solução do diagrama de montagem é encontrada usando aprendizado por reforço usando os algoritmos PLA e FALA.

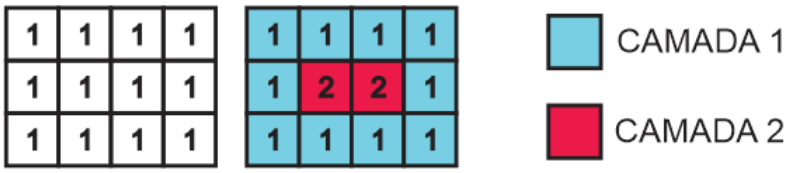
Observe que se existe um bloco de uma unidade em M_{ent} na posição $M_{ent}[1][1][1]$, então existe um autômato na matriz M_{aut} na posição $M_{aut}[1][1][1]$. O número total de autômatos é igual ao número total de blocos de uma unidade da matriz M_{ent}. Cada autômato possui um conjunto de ações com probabilidades correspondentes. Cada uma destas ações representa um tipo de bloco. Para realização da montagem das estruturas usando vários autômatos, os seguintes procedimentos devem ser definidos:

1. Os blocos só podem ser colocados na posição vertical ou horizontal;
2. Os blocos horizontais devem ser colocados com o seu ponto de origem a esquerda;
3. Os blocos verticais devem ser colocados com seu ponto de origem a baixo;
4. Os blocos dos níveis inferiores devem ser colocados primeiro que os superiores;
5. Os blocos devem ser classificados por camadas concêntricas M_C;
6. O mesmo bloco não pode estar em duas camadas concêntricas ao mesmo tempo e nem ocupar valores além dos limites de M_{ent};
7. Cada ação do autômato é referente a somente um tipo de bloco;
8. Os autômatos só podem ter ações que fazem parte do conjunto de blocos disponíveis para a montagem V_b.

Para definir a representação das ações é usada uma representação numérica para identificar ações de blocos horizontais e ações de blocos verticais. Os blocos horizontais são definidos por números inteiros positivos e os blocos verticais por números inteiros negativos. O valor do número inteiro refere-se ao número de unidades do bloco. Por exemplo, o bloco de duas unidades vertical é representado por -2 e o bloco de uma unidade é representado tanto por 1 ou -1 devido a sua simetria.

A montagem da estrutura é feita por camadas concêntricas. Para demonstrar como funciona a montagem em camadas concêntricas é usado como exemplo uma matriz M_{ent} unitária de dimensão 4x3x1, mostrada na Figura 39. A Figura 39 ilustra duas camadas concêntricas.

Figura 39 – Caso com camadas concêntricas.



Para encontrar o número total de camadas concêntricas de uma estrutura é usada a Eq. 5.1. A Eq. 5.4 informa qual camada concêntrica uma determinada posição (M_X, M_Y) faz parte.

$$M_{Ctotal} = ceil\left(\frac{min\left(\begin{bmatrix} M_X & M_Y \end{bmatrix}\right)}{2}\right) \tag{5.1}$$

$$M_C = min(\begin{bmatrix} M_X & M_Y & M_{Xtotal} - M_X & M_{Ytotal} - M_Y \end{bmatrix}) \tag{5.2}$$

onde, M_X e M_Y são as posições da matriz em algum nível de altura do eixo Z. M_{Ctotal} é o número total de camadas concêntricas e M_C é o número da camada concêntrica de uma determinada posição.

Situações em que a estrutura fica semelhante a Figura 36 (ver seção 4.5) são resolvidas adotando a montagem por camadas concêntricas. Nessa maneira a montagem é iniciada de uma camada concêntrica mais interna para uma mais externa.

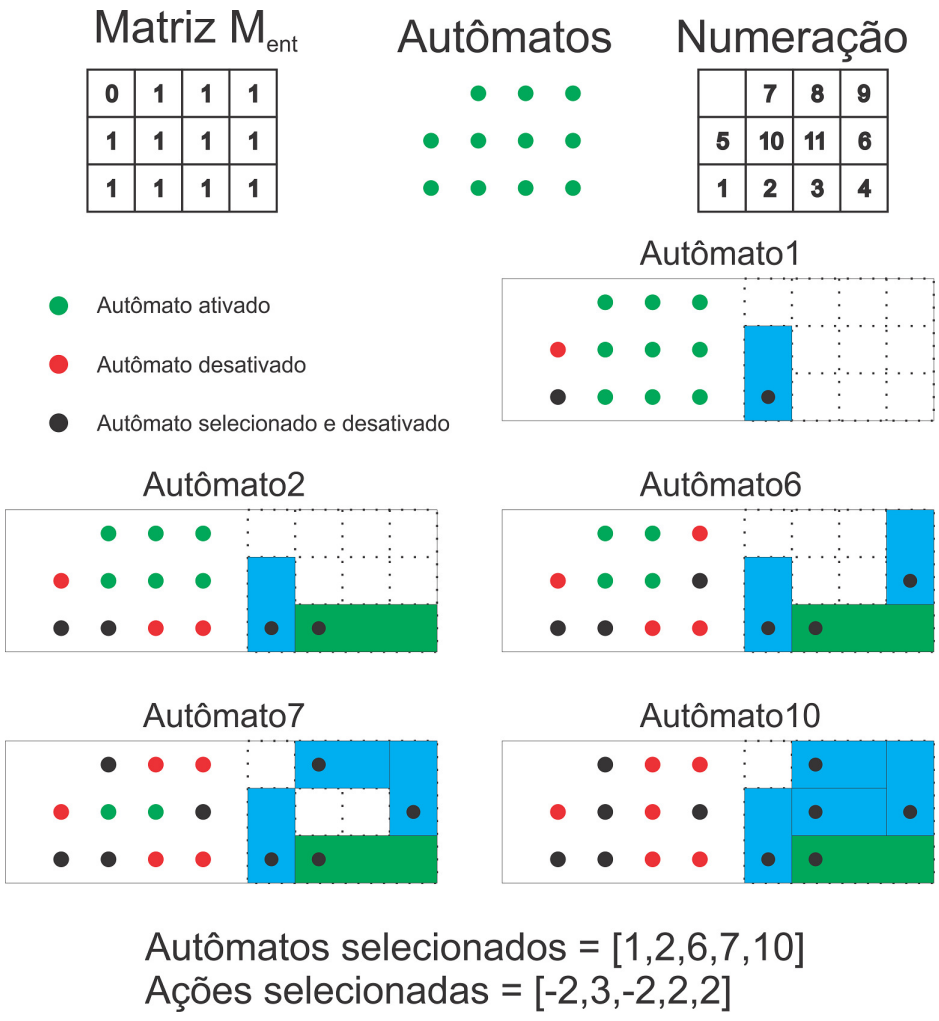
Usando blocos de uma, duas e três unidades é possível montar várias estruturas. Para isso deve-se colocar os blocos nos espaços que possuem o número 1 na matriz M_{ent}. Como a montagem é em camadas concêntricas o bloco de uma camada concêntrica não pode ocupar o espaço de outra camada concêntrica. Uma outra restrição que deve ser respeitada é a orientação dos blocos, que devem ser somente na vertical ou horizontal e com seus pontos de origem à esquerda e abaixo respectivamente.

Sabendo que na matriz M_{aut} existe um autômato na mesma posição de uma célula da matriz M_{ent}, esse autômato escolhe um determinado tipo de bloco, o qual é inserido na célula

de M_{ent} correspondente a posição do autômato. Quando um autômato seleciona uma ação, se o bloco for maior que uma unidade ele vai ocupar o espaço dos autômatos vizinhos.

A ordem de seleção dos autômatos e sua numeração é feita da camada concêntrica mais externa para mais interna, começando com o autômato mais à esquerda e abaixo e prosseguindo até o ultimo autômato da mesma linha, em seguida a seleção vai para a próxima linha. No início, todos os autômatos estão ativados e só são desativados quando escolhem uma ação ou quando o seu espaço é ocupado por um bloco escolhido pelo autômato vizinho. A estrutura é finalizada quando todos os autômatos estão desativados. A Figura 40 mostra um exemplo de montagem usando os autômatos.

Figura 40 – Estrutura gerada pelos autômatos.



A seleção da ordem de ativação dos autômatos é feita junto com a criação da matriz M_{aut} a função responsável pela sua criação é chamada de *GerarAutomata*.

A função *GerarAutomata* tem como entrada a matriz M_{ent} e o vetor de blocos disponíveis V_b. A função inicia-se com a geração da ordem de seleção dos autômatos. Essa seleção é realizada no primeiro nível de altura $M_Z = 1$ e na camada concêntrica mais externa $M_C = 1$. Para isso é feita uma leitura em toda a primeira linha $M_Y = 1$, se encontrado um autômato nessa linha,

ele é classificado como autômato de estado 1. Então caso existam mais autômatos na mesma linha, eles vão ser classificados em ordem crescente. Quando a leitura de autômatos da linha é finalizada a verificação passa para um valor seguinte de linha M_Y, o processo ocorre até que todas as linhas da camada concêntrica $M_C = 1$, tenham sido analisadas. Finalizando a verificação de $M_C = 1$ o algoritmo passa para o próximo valor de camada concêntrica M_C e o processo se repete até que todas as camadas concêntricas do primeiro nível (referente a altura da estrutura) tenham sido analisadas. Quando todas as camadas concêntricas do primeiro nível de altura ($M_Z = 1$) forem analisadas, o algoritmo começa novamente o processo em um valor seguinte de M_Z. Esse processo se repete até que toda matriz tenha sido analisada.

No final, a ordem de seleção é armazenada em cada autômato da matriz M_{aut}. Definido a ordem de seleção de cada autômato, deve-se analisar o conjunto de ações que cada autômato deve possuir.

A escolha do conjunto de ações começa com a seleção do primeiro autômato, realizando uma verificação na vizinhança. Cada tipo de bloco é um número inteiro positivo ou negativo, então é analisado se a posição da célula que o autômato está associado mais o valor absoluto do bloco gera alguma restrição. Se o bloco tem valor negativo é feito uma soma nos valores de M_Y, se positivo, nos valores de M_X. Caso o bloco ocupe mais de uma camada concêntrica ele não pode fazer parte do conjunto de ações do autômato. Com o auxílio da Eq. 5.4 é possível saber se o bloco é válido ou não como ação. Se o bloco ocupar um valor maior que os limites da matriz ((M_{Xtotal}, M_{Ytotal})) o bloco também não é válido como ação.

A probabilidade de cada ação inicialmente é igual a:

$$p_i = \frac{1}{k} \tag{5.3}$$

Onde k é o número total de ações do autômato. Para o método PLA, inicialmente o valor de u é igual ao vetor de probabilidade gerado.

O Algoritmo 1 tem a função de criar uma estrutura viável. Ela começa selecionando o primeiro autômato. Esse autômato escolhe uma ação baseada na sua distribuição de probabilidade. Essa ação gera um bloco que pode desativar um ou mais autômatos dependendo do tamanho do bloco selecionado. Em seguida, um outro autômato ativado é escolhido para selecionar uma ação e o processo continua até que a função desative todos os autômatos. Ao final desse processo é gerado uma lista de autômatos selecionados e uma lista de ações.

Algoritmo 1: Funções para gerar um diagrama de montagem válido

```
1  Função ColocaBloco
       Entrada : NivelAutomatos, estado
       Saída   : NovoEstado, ação
2      para M_Y ← 1 até M_Ytotal faça
3          para M_X ← 1 até M_Xtotal faça
4              se NivelAutomatos[M_X][M_Y].estado == estado então
5                  ação ← selecionarAção(NivelAutomatos[M_X][M_Y].prob)
6                  NovoEstado ← VerificarNovoEstado(NivelAutomatos, estado, ação)
7              fim
8          fim
9      fim
10 Função GerarEstrutura
       Entrada : M_aut
       Saída   : ListaAção, ListaEstado
11     para i ← 1 até M_Ztotal faça
12         NivelAutomatos ← M_aut[:][:][M_Z]
13         estado ← 1
14         i ← 1
15         enquanto estado[i] ≠ vazio faça
16             [ListaAção[M_Z][i], ListaEstado[M_Z][i+1]] ← ColocaBloco(NivelAutomatos, estado[i])
17             i ← i+1
18         fim
19     fim
```

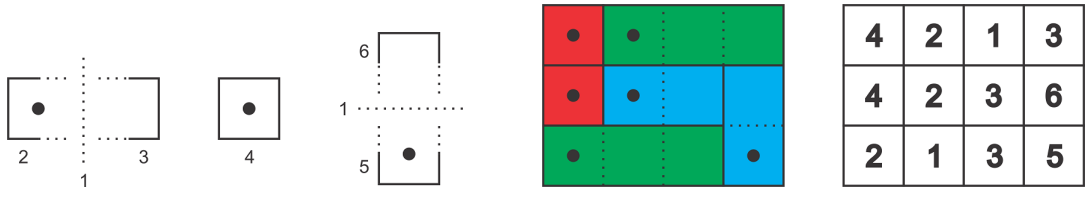
5.2.1 Função Custo do diagrama de montagem

Quando um diagrama é criado pelo Algoritmo 1 ele é avaliado. Essa avaliação é feita analisando o número de blocos definidos para a estrutura mais a quantidade de restrições que não foram atendidas. O algoritmo busca aprender um diagrama de montagem com o menor número de blocos, levando em conta as conexões entre os blocos nos níveis superiores ou inferiores.

A análise de restrição é feita através da verificação das conexões entre os blocos. Para saber se a montagem realizada possui alguma falha de conexão, devem ser analisados os limites esquerdo e direito de cada bloco da estrutura. Isso é feito codificando os limites e regiões centrais dos blocos em números.

A codificação é feita como mostrada na Figura 41. Os blocos horizontais têm seus limites esquerdos representados pelo número 2 e o limite direito pelo número 3, a região central é representado pelo número 1. Blocos de uma unidade são representados pelo número 4. Os blocos verticais possuem seu limite esquerdo representado pelo número 5, seu limite direito pelo número 6 e sua região central pelo número 1.

Figura 41 – Codificação de uma estrutura para identificar os limites esquerdos e direitos.



(a) (b)

Para saber se existe alguma falha de conexão é feita uma verificação na posição (M_X, M_Y, M_Z) com a posição $(M_X, M_Y, M_Z + 1)$. Caso uma mesma posição (M_X, M_Y) com um valor de nível de altura mais acima não possuir as combinações listadas a seguir, a estrutura tem boas conexões. As combinações são as seguintes:

$$\left[\begin{bmatrix} 4 & 2 \\ 5 & 4 \\ 3 & 3 \\ 3 & 4 \end{bmatrix} \quad \begin{bmatrix} 2 & 4 \\ 4 & 6 \\ 4 & 4 \end{bmatrix} \quad \begin{bmatrix} 4 & 3 \\ 6 & 4 \\ 5 & 5 \end{bmatrix} \quad \begin{bmatrix} 4 & 5 \\ 2 & 2 \\ 6 & 6 \end{bmatrix}\right] \tag{5.4}$$

Uma variável F_r guarda o número total de falhas de conexão devido às restrições envolvidas na geração do diagrama da estrutura. Para descobrir o número total de blocos usados na montagem (F_b), basta verificar o número de autômatos que foram selecionados para geração do diagrama de montagem.

A função de custo para geração do diagrama de montagem é mostrada na Eq 5.5. Essa função é o somatório das falhas (F_r) e número total de blocos (F_b). O método por aprendizado por reforço visa encontrar o menor valor de custo na criação de uma estrutura.

$$F_{rb} = F_r + F_b \tag{5.5}$$

5.2.2 Sinal de reforço e esquema de aprendizagem

A função de custo dada pela Eq. 5.5 retorna valores inteiros maiores que 0, mas nas abordagens de aprendizado usadas (FALA e PLA), o valor do sinal de reforço enviado pelo ambiente deve estar entre 0 e 1, então para limitar o reforço para ficar entre 0 e 1, é usada uma função de avaliação de desempenho para calcular o valor do reforço $b(t)$,conforme usada por Santos (2014), Santos, Givigi e Nascimento (2015) sendo escrita na Eq 5.6:

$$b(t) = min\left(max\left(0, \frac{J_{med} - J(t)}{J_{med} - J_{min}}\right), 1\right) \tag{5.6}$$

Usando esse avaliador de desempenho, o sinal $b(t)$ fica entre os valores de $[0, 1]$. Para que essa função funcione, ela precisa guardar valores de custo F_{rb}. Esses valores são guardados na variável J. A variável J funciona como uma memória e consegue armazenar uma capacidade igual a R valores. $J(t)$ é o valor atual de F_{rb} que já está armazenado em J. Se a memória de J possuir mais de R valores o valor de J $(R+1)$ é apagado. J_{med} é a média dos R valores de custo armazenados e J_{min} é o menor valor. Quando encontrado um valor maior que J_{med} a saída será igual a 0 e quando menor que J_{min} a saída será igual a 1, garantindo que somente bons resultados tenham atualização na probabilidade. Cada autômato da matriz M_{aut} possui um J para armazenar o custo da estrutura quando este autômato foi selecionado.

Com a lista de autômatos selecionados na geração do diagrama de montagem e as ações escolhidas, cada autômato selecionado, pode gerar um valor de recompensa usando o avaliador de desempenho. Em seguida são atualizadas as probabilidades usando os esquemas de atualização. Para o algoritmo FALA são usados as Eq. 2.3 e 2.4 e para o algoritmo PLA são usadas as Eq. 2.9 e 2.10.

Quando terminado o processo de atualização das probabilidades é então gerado um novo diagrama da estrutura. Esse diagrama é criado por autômatos que já tiveram suas probabilidades atualizadas. A próxima etapa é gerar um novo diagrama e calcular um novo valor de custo para esse diagrama. Em seguida são atualizados os valores de probabilidades dos autômatos selecionados na geração do diagrama. Esse processo se repete até um número máximo de iterações.

Em cada iteração do algoritmo, os valores de custo são analisados. Quando encontrado um valor de custo menor do que calculado anteriormente, os blocos e autômatos selecionados na geração do diagrama são armazenados para serem o resultado do melhor diagrama da estrutura encontrado. O Algoritmo 2 mostra todo o processo de aprendizagem.

Algoritmo 2: Atualização das probabilidades do diagrama de montagem.

```
1  Função Aprendizado
       Entrada : M_ent, metodo
       Saída   : MelhorListaAções, MelhorListaEstados
2      MenorCusto ← 100000000
3      M_aut ← GerarAutomatos(M_ent, V_b)
4      enquanto iterações ≤ MaxIterações faça
5          [ListaAções, ListaEstados] ← GerarEstrutura(M_aut)
6          Custo ← VerificarCusto(ListaAções, ListaEstados)
7          se Custo<MenorCusto então
8              MelhorListaAções ← ListaAções
9              MelhorListaEstados ← ListaEstados
10         fim
11         se Metodo == PLA então
12             M_aut ← AtualizaPLA(M_aut, Custo, ListaAções, ListaEstados)
13         fim
14         senão
15             M_aut ← AtualizaFALA(M_aut, Custo, ListaAções, ListaEstados)
16     fim
```

O diagrama de montagem deve possuir suas dimensões reais em relação a coordenada X_g, Y_g, Z_g. Para fazer a transformação da representação simplificada para coordenadas reais é necessário usar o valor da aresta do bloco de uma unidade (φ) e os valores de X_{offset} e Y_{offset}. Os valores de X_{offset} e Y_{offset} são usados para mover os blocos do diagrama para alguma posição do ambiente. O diagrama de montagem é escrito usando os valores dos estados dos autômatos e de suas melhores ações. O diagrama de montagem é chamado de D_A e a Tabela 1 mostra a sua estrutura.

Tabela 1 – Modelo de um diagrama de montagem

I_A	Posição X	Posição Y	Posição Z	Orientação	Tipo de Bloco	Camada
1	$M_{X,1}\varphi\varphi - \varphi/2 + X_{offset}$	$M_{Y,1}\varphi\varphi - \varphi/2 + Y_{offset}$	$M_{Z,1}\varphi - \varphi/2$	θ_1	$B_{tipo,1}$	$M_{C,1}$
2	$M_{X,2}\varphi\varphi - \varphi/2 + X_{offset}$	$M_{Y,2}\varphi\varphi - \varphi/2 + Y_{offset}$	$M_{Z,1}\varphi - \varphi/2$	θ_2	$B_{tipo,2}$	$M_{C,2}$
⋮	⋮	⋮	⋮	⋮	⋮	⋮
n_d	$M_{X,n_d}\varphi\varphi - \varphi/2 + X_{offset}$	$M_{Y,n_d}\varphi\varphi - \varphi/2 + Y_{offset}$	$M_{Y,n_d}\varphi - \varphi/2$	θ_{n_d}	B_{tipo,n_d}	M_{C,n_d}

O diagrama de montagem possui 6 colunas e o número de linhas iguais ao número de blocos necessários para construir a estrutura. A primeira coluna é uma identificação do bloco (I_A), de forma numérica representa o bloco com valores de 1 até o número total de blocos, definido como n_d. A segunda coluna informa o valor de posição do ponto de origem do bloco no ambiente em relação ao eixo X_g, a terceira coluna informa o valor de M_Y do ponto de origem do bloco no ambiente em relação ao eixo Y_g, a quarta coluna informa o valor do ponto de origem do bloco no ambiente em relação ao eixo Z_g, a quinta coluna informa se o bloco é horizontal (0º) ou vertical (90º), a sexta coluna informa o tipo de bloco e a sétima coluna informa de qual camada concêntrica de montagem pertence o bloco.

O valor do identificador I_A não se repete no diagrama de montagem, então cada bloco do diagrama possui somente um valor. Quando é preciso ter informações de algum bloco do diagrama de montagem como, a posição ou orientação, é usado o identificador I_A para encontrar os dados no diagrama.

5.3 GERAÇÃO DO PLANO DE EXECUÇÃO

A geração do plano de execução é feita com o conhecimento do diagrama de montagem (D_A) e de uma lista de locais de manipulação (L_A). O plano de execução é uma sequência ótima de procedimentos de manipulação e de montagem que visa diminuir o tempo de operação do robô para a montagem da estrutura.

A lista L_A possui uma composição semelhante ao diagrama de montagem D_A. Nesta lista é informada a identificação do local de manipulação, a posição do local do bloco, a sua orientação, a sua altura e o seu o tipo de bloco. A informação da camada concêntrica de montagem não é usada em L_A.

Para realizar a geração do plano de execução é preciso definir algumas restrições:

1. A montagem deve ser realizada de uma camada concêntrica mais interna para uma mais externa;

2. A montagem só deve ir para um nível de altura superior, quando todos os blocos do nível de altura inferior estiverem posicionados;

3. Não existem limites nas quantidades de blocos em um mesmo local de manipulação.

A tarefa de montagem das estruturas é definida pela seleção, a partir do processo de aprendizado, dos identificadores no diagrama de montagem, chamado de I_A. Com o identificador I_A é possível obter informações da posição, da orientação e do tipo de bloco que deve ser montado. Após a escolha de um identificador I_A é feita a verificação dos locais de manipulação que possuem o tipo de bloco do identificador I_A escolhido. Essa etapa é feita através da seleção do identificador na lista de locais de manipulação, chamado de I_M que tenha o tipo de bloco escolhido.

Ao escolher o local de manipulação, o robô executa o procedimento de manipulação. O robô desloca-se da sua posição, em direção a posição do bloco da lista L_A para capturá-lo. Ao finalizar o procedimento de manipulação, o robô executa um procedimento de montagem para colocar o bloco na posição e na orientação que foi informado pelo diagrama de montagem. Esse processo ocorre até que todos os blocos do plano de montagem tenham sido posicionados. As trajetórias realizadas pelo robô nos procedimentos de manipulação e de montagem são geradas por um algoritmo A*.

O processo de geração do plano de execução é feito usando uma rede de autômatos que tem suas probabilidades atualizadas através do algoritmo PLA. A rede de autômatos é composta por dois tipos de autômatos, os autômatos de montagem e de manipulação. Os autômatos de montagem, são autômatos que possuem ações iguais aos números de identificadores de um nível de altura e uma camada concêntrica específica no diagrama de montagem. Os autômatos de manipulação são autômatos que possuem um número de ações iguais ao número de locais de manipulação de um determinado tipo de bloco.

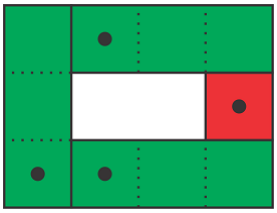
A Tabela 2 mostra um diagrama de montagem e a lista de locais de manipulação da estrutura mostrada na Figura 42a. Esse diagrama de montagem possui somente nível de altura, uma camada concêntrica, $\varphi = 8cm$, $X_{offset} = 100cm$ e $Y_{offset} = 100cm$

Tabela 2 – Exemplo de um diagrama de montagem e de uma lista de locais de manipulação

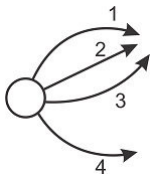
Diagrama de Montagem						
I_A	Posição X (cm)	Posição Y (cm)	Posição Z (cm)	Orientação	Tipo de Bloco	Camada
1	100	100	4	90º	3	1
2	108	100	4	0º	3	1
3	128	108	4	0º	1	1
4	108	120	4	0º	3	1

Lista de locais de Manipulação						
I_M	Posição X (cm)	Posição Y (cm)	Posição Z (cm)	Orientação	Tipo de Bloco	Camada
1	200	200	4	45º	3	-
2	200	20	4	315º	1	-
3	20	200	4	225º	3	-

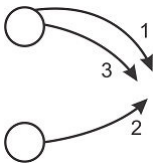
Figura 42 – Exemplo de uma estrutura e seus autômatos de montagem e de manipulação.



(a) Estrutrua



(b) autômato de montagem

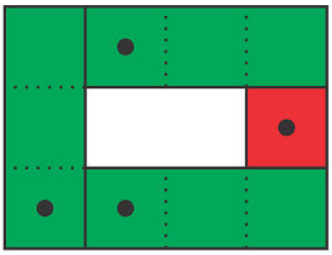


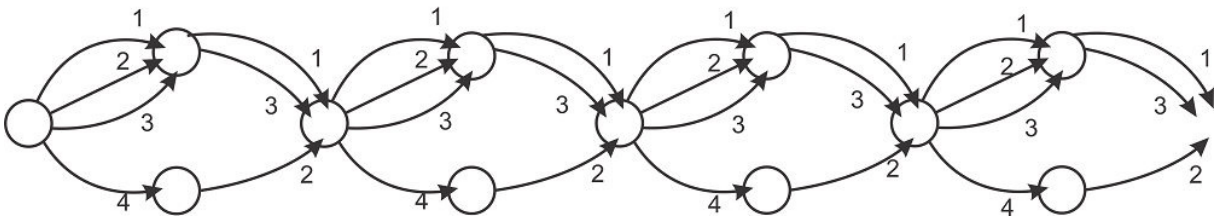
(c) autômato de manipulação

A Figura 42b mostra um autômato de montagem com 4 ações, cada ação representa um identificador do diagrama de montagem. A Figura 42c mostra dois autômatos de manipulação. O autômato referente ao bloco de 3 unidades possui 2 ações, cada ação é um identificador da lista dos locais de manipulação. O segundo autômato de manipulação é referente ao bloco de 1 unidade, como na lista de locais de manipulação existe somente um local de manipulação para o bloco de uma unidade, o autômato possui somente uma ação.

A ligação entre os autômatos é feita a partir das ações. A ligação é feita de um autômato de montagem para um autômato de manipulação e de um autômato de manipulação para um autômato de montagem. Esse processo continua até que o diagrama tenha sido finalizado. A Figura 43 mostra a conexão entre os autômatos do diagrama de montagem da Tabela 2.

Figura 43 – Representação de uma rede de autômatos para geração do plano de execução.



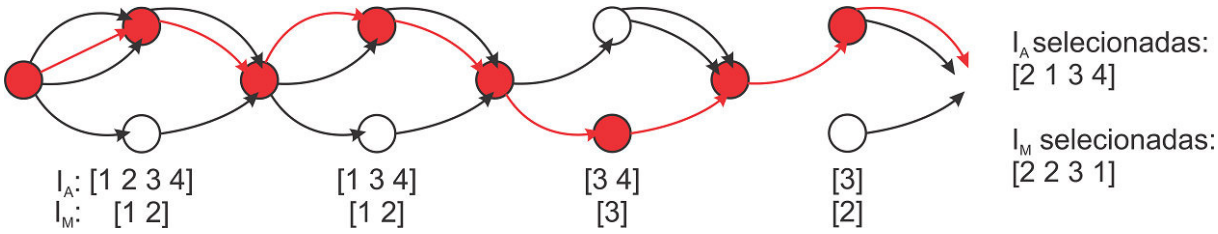


Percebe-se que uma ação que representa um identificador de um bloco de três unidades do diagrama de montagem, conectam-se com autômatos de manipulação referentes aos blocos de três unidades. O mesmo acontece a ação do bloco de uma unidade.

Quando uma ação com um certo identificador do diagrama de montagem é escolhida, o autômato de montagem seguinte desativa de seu conjunto de ações a ação referente ao identificador já escolhido. Isso é feito para que no final todas as ações criem um plano de execução válido.

Se existir um identificador repetido é como se fosse selecionado a montagem de um bloco no mesmo lugar mais de uma vez. A Figura 44 mostra a seleção das ações sem a repetição das posições.

Figura 44 – A seleção de ações de um plano de execução



No início, a probabilidade de cada ação é igual a:

$$p_i = \frac{1}{k} \tag{5.7}$$

onde k é o número de ações do autômato escolhido.

Quando uma ação é desativada, as ações restantes devem ter suas probabilidades distribuídas para que a soma fique igual a 1. Essa distribuição é feita com uma variável de probabilidade auxiliar. A probabilidade auxiliar é usada somente no momento da escolha de uma ação e quando as outras estão desativadas. A distribuição é feita dividindo a probabilidade do autômato pela soma das probabilidades dos autômatos ativados. Seja um autômato com uma probabilidade p_1 de um conjunto de probabilidades igual a $[p_1, p_2, p_3, p_4]$. Se a ação referente a probabilidade p_4 foi desativada, então a nova probabilidade p_1 é igual a:

$$p_{n1} = \frac{p_1}{soma([p_1, p_2, p_3])} \tag{5.8}$$

Em uma camada concêntrica especifica, cada nível de altura da estrutura possui uma rede semelhante ao da Figura 44. Quando é selecionado a ação do último autômato de manipulação desse nível de altura, essa ação faz uma ligação com o primeiro autômato do nível de altura superior e da mesma camada concêntrica. Esse processo acontece até chegar no último nível de altura. Chegando no último nível de altura, a ação leva a ativação de um autômato do primeiro nível de altura da camada concêntrica seguinte e esse processo acontece até terminar o plano de montagem. O Algoritmo 3 mostra a criação dos autômatos para geração do plano de execução.

Algoritmo 3: Geração dos autômatos para o plano de execução

```
1  Função CriaAutomatoPlano
       Entrada : D_A, L_A
       Saída   : AutomatoMontagem, AutomatoManipulação
2      L_A ← ordenar L_A coluna das camadas e na coluna dos níveis
3      D_A ← ordenar D_A coluna das camadas e na coluna dos níveis
4      BlocosDiferentes ← tipos de blocos usados na montagem
5      para i ← 1 até tamanho de D_A faça
6          AutomatoMontagem[i].ações ← todas as I_A com a camada e nivel igual da D_A[i][1]
7          AutomatoMontagem[i].prob ← 1/(tamanho de AutomatoMontagem[i].ações)
8          AutomatoMontagem[i].u ← AutomatoMontagem[i].prob
9          para j ← 1 até tamanho de BlocosDiferentes faça
10             AutomatoManipulação[i][j].ações ← todas as I_M com tipo de bloco igual a L_A[j][1]
11             AutomatoManipulação[i][j].prob ← 1/(tamanho de AutomatoManipulacao[i][j].ações)
12             AutomatoManipulação[i].u ← AutomatoManipulação[i].prob
13         fim
14     fim
```

Para gerar um plano de execução válido é usado a função *PlanoValido*. Nessa função são usados os autômatos gerados pelo Algoritmo 3. O pseudocódigo é mostrado no Algoritmo 4.

Algoritmo 4: Geração de um plano de execução válido.

```
1  Função SelecionaAção
       Entrada : ProbAutomato
       Saída   : I
2      se Alguma ação já foi selecionada então
3          para i ← 1 até tamanho de ações não selecionadas faça
4              AuxProb[i]← (probabilidade da ação selecionada) /soma(probabilidade das ações ativadas)
5          fim
6      fim
7      senão
8          AuxProb[i]← ProbAutomato
9          I ←SelecionarAção(AuxProb)

10 Função PlanoValido
       Entrada : D_A, AutomatoMontagem, AutomatoManipulação
       Saída   : PlanoExecução
11     j ← 1
12     para i ← 1 até tamanho de D_A faça
13         AçãoMontagem← SelecionaAção(AutomatoMontagem[i].prob)
14         TipoBloco← VerificarTipoBloco(AcaoMontagem,D_A)
15         AcaoManipulação← SelecionaAção(AutomatoMontagem[i][TipoBloco].prob)
16         PlanoExecução[j]← AçãoManipulação
17         PlanoExecução[j+1]← AçãoMontagem
18         j ← j+2
19     fim
```

A saída do algoritmo 4 é um vetor com uma sequência dos identificadores I_A e I_M que codificam o plano de execução. Essa é uma maneira compacta e simples de se gerar um plano de execução. A estrutura da Figura 44 possui uma sequência igual a $[2,2,2,1,3,3,1,4]$.

Essa é uma sequência de procedimentos de manipulação e de montagem. As posições dos blocos são obtidas através dos identificadores I_A e de I_M informados na sequência.

5.3.1 Função custo do plano de execução

O plano de execução é ótimo se conseguir chegar a melhor sequência de procedimentos de manipulação e de montagem, que resulte na menor distância percorrida. Para verificação

da distância percorrida é usado o algoritmo A*. A entrada do A* é a posição atual do robô no ambiente e a posição de $I_M = 1$ como destino. A saída do algoritmo A* é o deslocamento total sem colisão do robô, do procedimento de manipulação, que é armazenado em uma variável F_m.

O próximo passo é usar a posição de manipulação com identificador $I_M = 1$ como posição inicial e a posição de identificador $I_A = 1$ como posição de destino. O retorno é o deslocamento total do robô, do procedimento de montagem, que é armazenado na variável F_a. Na próxima etapa é informado para o algoritmo A* que existe um bloco no ambiente(novo obstáculo) na posição definida no diagrama de montagem pelo identificador $I_A = 1$. Esse processo continua até que tenha sido executado todo o plano de execução gerado. No final é possível obter o somatorio de deslocamento total do robô na execução do plano, armazenado na variável F_{ma}. A função de custo é dada pela Eq. 5.9. O Algoritmo 5 mostra como é feito esse processo.

$$F_{ma} = \sum (F_m + F_a) \tag{5.9}$$

Algoritmo 5: Processo de aprendizagem.

```
1  Função CalculaCusto
       Entrada : PlanoExecução
       Saída   : F_ma
2      F_a ← 0
3      F_m ← 0
4      F_ma ← 0
5      início ← Posição inicial do robô
6      para i ← 1 até tamanho de PlanoExecução faça
7          se i for multiplo de 2 então
8              início ← destino
9              destino ← posição do bloco com identificação igual a PlanoExecução[i]
10             [distância, obstáculo] ← A*(início, destino, obstáculo)
11             F_a ← F_a+distância
12         fim
13         senão
14             destino ← posição do bloco com identificação igual a PlanoExecução[i]
15             [distância, obstáculo] ← A*(início, destino, obstáculo)
16             F_m ← F_m+distância
17     fim
18     F_ma ← F_m + F_a
```

5.3.2 Geração de um plano de execução

A função de custo dada pela Eq. 5.9 retorna valores maiores que 1 então é usado uma função de avaliação de desempenho (Eq. 5.6) para gerar recompensas entre 0 e 1. Isso faz com que cada autômato desse processo tenha uma variável de memória J para guardar os valores de custo.

O Algoritmo 4 gera um plano de execução válido que é avaliado pela função de custo. Cada autômato que selecionou uma ação guarda o custo e gera sua própria recompensa. O valor das probabilidades são atualizadas através do algoritmo PLA, feitas pelas Eq. 2.9 e 2.10.

Quando as probabilidades são atualizadas um novo plano de execução é gerado e seu custo avaliado. Esse processo continua até um valor máximo de iterações. O valor de custo

é comparado com o menor valor de custo armazenado, se o valor atual for menor que o já armazenado então esse plano é melhor que o anterior. O melhor plano fica armazenado e é atualizado sempre que encontra um plano com custo menor, no final das iterações o plano armazenado é o plano ótimo de execução. O Algoritmo 6 mostra como é feito esse processo.

Algoritmo 6: Processo de aprendizagem.

```
1  Função Aprendizado
       Entrada : D_A, L_A
       Saída   : MelhorPlanoExecução
2      [AutomatoMontagem, AutomatoManipulação]← CriaAutomatoPlano(D_A, L_A)
3      MenorCusto ← 1000000000
4      enquanto iterações ≤ MaxIterações faça
5          PlanoExecução ← PlanoValido(D_A, AutomatoMontagem, AutomatoManipulação)
6          Custo ← CalculaCusto(PlanoExecução)
7          se Custo < MenorCusto então
8              MelhorPlanoExecução ← PlanoExecução
9          fim
10         [AutomatoMontagem, AutomatoManipulação] ← AtualizaPLA(AutomatoMontagem, AutomatoManipulação)
11     fim
```

O plano de execução é reescrito em forma de procedimentos de montagem e manipulação. Esse processo é feito usando o *MelhorPlanoExecução* e aplicando o algoritmo A* para gerar trajetórias para os procedimentos de montagem e manipulação. As trajetórias geradas pelo A* evitam colisões com os blocos (obstáculos) que são colocados no ambiente. Dessa maneira o robô consegue executar todos os comandos.

6 RESULTADOS

Este capítulo apresenta os resultados dos algoritmos para geração do diagrama de montagem e para o planejamento da execução da tarefa. O capítulo inicia-se com o modo de escolher os melhores parâmetros para geração do diagrama de montagem. Inicialmente, os parâmetros são testados na estrutura do tipo cisterna, pois o valor de custo mínimo do diagrama de montagem é conhecido. Em seguida foram feitos o aprendizado do planejamento de execução variando o valor da memória J, com o objetivo de saber o seu efeito na obtenção de plano durante a fase de aprendizado.

Os algoritmos baseados na abordagem PLA e na abordagem FALA são usados para gerar diagramas de montagens das estruturas propostas no capítulo 3. A partir do diagrama de menor custo é usado o algoritmo de abordagem PLA para gerar o plano de execução.

Com o plano de execução gerado pelo algoritmo PLA, o robô simulado consegue executar os procedimentos de montagem e de manipulação. A demonstração do processo de construção é realizada por meio de ilustrações obtidas ao longo do tempo.

O plano de execução gerado para a construção de uma parede é testado experimentalmente usando um robô real. Na seção 6.3 são mostradas as etapas necessárias para que o robô real consiga montar a estrutura.

Os algoritmos PLA e FALA são executados em um processador Intel i5 de 1.7 GHz com 6 GB de memória RAM no *software* MATLAB.

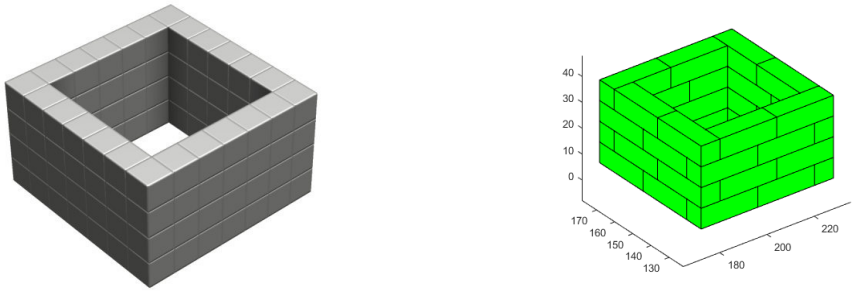
6.1 ANÁLISE DA ESCOLHA DOS PARÂMETROS

Nessa seção são usados dois algoritmos para geração do diagrama de montagem, um baseado na abordagem PLA e o outro baseado na abordagem FALA. A geração do plano de execução é feita usando somente o algoritmo baseado na abordagem PLA.

Para realização dos testes de aprendizado, foi escolhida uma estrutura de armazenamento de água, uma cisterna. A estrutura é composta por 4 paredes e possui 4 níveis de altura. A estrutura foi criada no *software* MagicalVoxel e sua forma final é vista na Figura 45a. A partir do modelo criado no MagicalVoxel é possível obter a matriz M_{ent}, que é usada como de entrada para o algoritmo de geração do diagrama de montagem.

No ambiente simulado são disponibilizados blocos de três, de duas e de uma unidade. Um usuário é capaz de gerar um diagrama de montagem da cisterna usando somente blocos de 3 unidades. Com o intuito de minimizar a função de custo da Eq. 5.5 os blocos podem ser organizados para zerar o número de falhas de conexão, dessa maneira o valor do custo é igual a zero falhas de conexão mais 32 blocos, totalizando em um valor igual a 32. Esse valor é o menor valor de custo possível na geração do diagrama de montagem da estrutura da cisterna. Na Figura 45b é mostrada a estrutura montada a partir do diagrama de montagem de custo igual a 32.

Figura 45 – Entrada do algoritmo de geração do diagrama de montagem e sua saída.



(a) Entrada do algoritmo (b) Estrutura de menor custo

Para o algoritmo de geração do diagrama de montagem são usados três critérios de parada do algoritmo:

- Número de iterações: Esse critério de parada é atingido quando o número de iterações chega a um valor máximo definido. Nessa situação, o algoritmo do diagrama de montagem retorna como resposta o valor mínimo de custo;
- Custo mínimo: Outro critério de parada é verificar se nas últimas 100 amostras de custo, foram encontradas um determinado valor de amostras com o custo mínimo. Esse processo é realizado a cada iteração e o número de amostras é definido pelo usuário;
- Valor de convergência: Por fim, o último critério é feito analisando a maior porcentagem do vetor de probabilidade de todos os autômatos ativados na criação do diagrama de montagem. Para isso é realizado o somatório dessa porcentagem e, em seguida, ela é dividida pelo número de autômatos ativados. O valor gerado é a média das probabilidades, sendo que esse valor tende a 1. Quando o algoritmo chega a um valor pelo usuário, o processo de aprendizagem é finalizado.

Para o aprendizado do diagrama de montagem são usados os seguintes valores para os critérios de parada: a) iteração máxima de 10000 iterações; b) custo mínimo igual a 32; c) valor da convergência é igual a 0.95.

A saída do algoritmo de geração do diagrama de montagem é um diagrama de montagem com um baixo valor custo. Esse diagrama de montagem é a entrada para o algoritmo do plano de execução. Diferentemente do diagrama de montagem, realizar a geração do plano de execução também é uma tarefa difícil para ser planejada pelos usuários. O critério de parada usado no algoritmo de geração do plano de execução é o número iterações. O valor máximo adotado foi de 3000 iterações.

6.1.1 Diagrama de montagem usando o PLA

O algoritmo de geração do diagrama de montagem usando o algoritmo PLA atualiza os valores do vetor u a partir das Eq. 2.9 e 2.10. Essas duas equações dependem de seis parâmetros, eles são: a variável responsável pelo limite positivo ou negativo do valor das componentes do vetor u, chamado de L; as variáveis responsáveis por manter o vetor u com seus valores dentro do limite L, chamados de K e n; a sequência de variáveis aleatórias i.i.d., chamada de s_i; a taxa de aprendizado λ_1 e a memória J do autômato. Para escolher qual o melhor valor de cada parâmetro são escolhidos alguns valores para alguns desses parâmetros. Os valores de cada parâmetro são:

- $K = 0.5; K = 1; K = 2$
- $L = 1; L = 5; L = 10$
- $n = 1; n = 2; n = 3$

Os outros parâmetros do algoritmo PLA possuem somente um valor. A taxa de aprendizagem $\lambda_1 = 0.1$, o $s_i = 0.1$ e a memória $J = 100$ dos autômatos.

Para analisar os efeitos que os valores de K, L e n fazem no número de iterações para encontrar o resultado de menor custo de uma cisterna, devem ser realizados 10 aprendizados para cada uma das combinações de parâmetro. As combinações de valores de parâmetros mais relevantes são mostrados na Tabela 3. Os dados da tabela são: a média de iterações do algoritmo até atingir algum critério de parada; a razão entre a quantidade de vezes que o algoritmo chegou no menor valor de custo (igual a 32) e o número total de treinamentos, chamado de taxa de acerto.

Tabela 3 – Aprendizados do diagrama de montagem para cada parâmetro K, L e n

K	n	L	média de iterações	taxa de acertos
1	1	5	4315.2	100%
2	3	10	4315.2	100%
0,5	1	5	4315.2	100%
0,5	3	10	4315.2	100%
1	2	1	10000	80%
2	3	1	10000	90%
1	1	1	10000	10%

Os aprendizados em que foram usados o valor de $L = 1$ tiveram o maior número de iterações, enquanto nos outros aprendizados com valores de L maior que 1, os resultados foram melhores. Para os próximos aprendizados do diagrama de montagem usando o algoritmo PLA, foram escolhidos os seguintes valores de parâmetros: $K = 1$, $L = 10$ e $n = 1$.

Outro parâmetro que é analisado é a memória J. A partir dos aprendizados com diferentes valores de memória J pretende-se verificar os efeitos no número de iterações para encontrar o

menor valor de custo. Semelhante ao teste dos valores K, L e n foram feitos 10 treinamentos para cada valor de memória J. O resultado obtido e os valores de J são vistos na Tabela 4. Essa tabela mostra a média de iterações até chegar em algum critério de parada e a taxa de acerto do algoritmo para atingir o menor valor de custo.

Tabela 4 – Tabela do PLA com os resultados da variação da memória.

memória J	média de iteração	taxa de acerto
5	4173.7	100%
10	4133.6	100%
50	4275.4	100%
70	4180.9	100%
100	4315.2	100%
200	4540.2	100%
500	4925.1	100%
1000	5197.4	100%
2000	5805.3	100%

Valores baixos de memória J, como o valor de memória igual a $J = 5$, faz com que os autômatos tenham poucos valores de custos anteriores armazenados. Dessa maneira usando a Eq. 5.6 valores obtidos abaixo de J_{med} vão possuir valores próximos ou igual a 1. Os valores baixos de memória J aceleram o processo de aprendizado, mas em algumas estruturas pode ser que não seja possível encontrar o de menor custo.

Os valores muito altos de memória como 1000 geram valores de recompensa muito perto de zero para baixos custos, o que acaba gerando uma demora na diminuição da curva de custo. Os valores medianos conseguem ter bons resultados pois, as recompensas não são nem tão altas nem tão baixas. O valor de memória J igual a 100 é suficiente para fazer o algoritmo encontrar uma boa solução para várias estruturas.

6.1.2 Diagrama de montagem usando o FALA

O algoritmo de geração do diagrama de montagem usando a abordagem FALA atualiza os valores de probabilidade p usando as Eq. 2.1 e 2.1. Diferentemente do algoritmo PLA, o algoritmo FALA possui somente dois parâmetros: a memória J e a taxa de aprendizado λ_1.

Para escolher o melhor valor de memória J foi usada a estrutura da cisterna para realizar os aprendizados. Nesses aprendizados foram escolhidos alguns valores de J mantendo o valor de $\lambda_1 = 0.1$ fixos. Foram realizados 10 treinamentos para cada valor de memória J. Os valores de J, os resultados da média de iterações até satisfazer um dos critérios de parada e a taxa de acerto são mostrados na Tabela 5.

Tabela 5 – Tabela do FALA com os resultados da variação da memória.

memória	média de iteração	taxa de acerto
5	1868,8	80%
10	1201	80%
50	1789,4	80%
70	1418,6	80%
100	1402,4	100%
200	1547,4	100%
500	1593,6	100%
1000	1708,8	100%

Assim como no algoritmo PLA, valores muito baixos ou muito altos de memória J não são indicados. Nesse caso os valores de memória J baixos acabaram fazendo a taxa de acerto do algoritmo diminuir. Quando o valor chega em $J = 100$, o algoritmo atinge o menor custo em todos os 10 aprendizados.

6.1.3 Comparação entre as abordagens de geração do diagrama de montagem

Os dois algoritmos de geração do diagrama de montagem possuem abordagens diferentes, então, foi necessário desenvolver uma maneira de comparar esses dois algoritmos.

Para realizar a comparação dos dois algoritmos, são usados os melhores valores dos parâmetros encontrados nas seções 6.1.1 e 6.1.2, em conjunto com a variação do valor da taxa de aprendizado. Os melhores valores usados para a geração do diagrama de montagem na estrutura da cisterna para a abordagem PLA foram: $K = 1$, $L = 10$, $n = 1$, $s_i = 0.1$ e J = 100. Por sua vez, para geração do diagrama de montagem usando a abordagem FALA o melhor valor de: $J = 100$. Os valores utilizados na taxa de aprendizado nos dois algoritmos são valores que começam em 0.005 e terminam em 0.9.

Para cada valor de taxa de aprendizado foram realizados 100 aprendizados e, com o conhecimento do menor custo do diagrama de montagem da cisterna, foi calculada a razão entre o número de vezes que o algoritmo chegou no menor valor de custo e o número total de aprendizados, chamado de taxa de acerto. A Tabela 6 mostra o número de iterações médias e o custo médio dos algoritmos PLA e FALA ao satisfazer algum critério de parada.

Com os resultados obtidos, percebe-se que aumentando a taxa de aprendizado o algoritmo começa a ter uma baixa taxa de acerto. Sendo que o FALA chega a 0% de acertos em altas taxas de aprendizado, enquanto no PLA o algoritmo continua tendo acertos mesmo em altas taxas de aprendizado.

Percebe-se pela Tabela 6 que o algoritmo FALA é mais suscetível a mudança da taxa de aprendizado, enquanto o algoritmo PLA a diferença é mais suave. Dessa maneira os dois algoritmos são bons para resolver o sistema de construção, mas deve-se ter cuidado na escolha da taxa de atualização.

Tabela 6 – Comparativo dos métodos com diferentes taxas de aprendizagem.

	PLA			FALA		
Taxa de aprendizado	Iterações médias	Custo médio	acertos (%)	Iterações médias	Custo médio	acertos (%)
0.005	10000	55.37	0	10000	43.37	0
0.01	10000	49.78	0	10000	32.11	93
0.02	10000	38.51	0	6333.9	32	100
0.03	10000	32.64	58	4218.3	32	100
0.06	7055	32	100	2150.1	32	100
0.08	5349	32	100	1905.6	32.39	95
0.1	4415.1	32	100	1504.2	32.21	96
0.15	2998.8	32	100	2068.6	33.14	76
0.2	2333.1	32	100	3343.7	33.59	59
0.25	2086.3	32.01	99	3301.2	34.88	38
0.35	1974.1	32.04	98	4509.1	36.39	22
0.4	1766.3	32.07	97	5489	37.75	7
0.45	1454.6	32.11	97	5220.1	38.55	3
0.5	1847.9	32.22	94	5233.3	39.89	2
0.6	1845.5	32.46	90	4346.2	42.37	0
0.7	2027.3	32.89	80	5756.9	43.74	1
0.8	1867.3	32.91	78	6112.1	45.18	0
0.9	1624.5	33.8	65	6736.2	46.67	0

Os algoritmos baseados PLA e FALA devem possuir parâmetros que garantam a solução do diagrama de montagem em um baixo valor de número de iterações e com uma alta taxa de acertos, ou seja, conseguir chegar no menor valor de custo possível. Os melhores resultados são obtidos com os parâmetros usados com uma taxa de acerto maior que 95%.

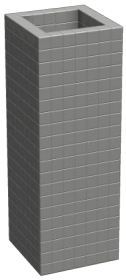
Para o algoritmo FALA encontrar o resultado com o menor número de iterações possíveis e com uma taxa de acerto maior que 95%, a taxa de aprendizado foi igual a 0.06. Se o número elevado de iterações não for um problema importante é possível escolher parâmetros que possuam altas taxas de acerto. Para esse caso é usado o valor de taxa de aprendizado igual a 0.1. Baseado na Tabela 5, o algoritmo FALA pode ter o número de iterações reduzido se usar um valor de memória $J = 5$, mas esse valor diminui a taxa de acerto. Usando o valor de memória $J = 100$ o algoritmo consegue ter uma alta taxa de acerto.

Para o algoritmo PLA encontrar o resultado no menor número de iterações possível com uma taxa de acerto maior que 95%, a taxa de aprendizado foi igual a 0.45. Se o número elevado de iterações não for um problema importante é possível escolher parâmetros que possuam altas taxas de acerto, para esse caso é usado o valor de taxa de aprendizado igual a 0.2. Baseado na Tabela 4, o algoritmo PLA pode ter o número de iterações reduzidas se usar um valor de memória $J = 5$, mas esse valor diminui a taxa de acerto. Usando o valor de memória $J = 100$ o algoritmo consegue ter uma alta taxa de acerto.

Para verificar a eficiência da geração de um diagrama ótimo de montagem foram realizados aprendizados, em uma estrutura com prolongamento horizontal e em uma estrutura com prolongamento vertical. As estruturas geradas no MagicalVoxel são mostradas na Figura 46.

Figura 46 – Estruturas com prolongamento vertical e horizontal.

(a) Prolongamento horizontal



(b) prolongamento vertical

Para os algoritmos FALA e PLA são usadas duas configurações de parâmetros, uma configuração rápida e outra configuração segura. Na configuração rápida, são escolhidos valores para os parâmetros em que o resultado é obtido em um baixo número de iterações, sem ter uma alta taxa de acerto. Na configuração segura o número de iterações não é uma preocupação, então são escolhidos valores com alta taxa de acerto. Os parâmetros foram escolhidos baseados nos resultados da geração do diagrama de montagem para a estrutura da cisterna.

Para o PLA os valores dos parâmetros para as configurações são:

- Seguro: $K = 1$, $L = 10$,$n = 1$, $\lambda = 0.2$, $s_i = 0.1$ e $J = 100$
- Rápido: $K = 1$, $L = 10$,$n = 1$, $\lambda = 0.45$, $s_i = 0.1$ e $J = 10$

Para o FALA os valores dos parâmetros para as configurações são:

- Seguro: $\lambda = 0.06$ e $J = 100$
- Rápido: $\lambda = 0.1$ e a $J = 10$.

Foram feitos 10 aprendizados com um número máximo de iterações iguais a 75000. Os resultados do aprendizado das duas estruturas podem ser vistos na Tabela 7.

Tabela 7 – Tabela de resultados das estruturas com prolongamento horizontal e vertical.

	Estrutura Horizontal					Estrutura Vertical				
	custo médio	custo mínimo	iterações	tempo	mínimo	custo médio	custo mínimo	iterações	tempo	mínimo
PLA seguro	167.3	160	36796	58 min	4	113.4	112	24417	24 min	3
PLA rapido	161.8	160	35126	56 min	8	115	112	36586	37 min	1
FALA seguro	168.6	160	21232	27 min	5	114.8	112	92763	77 min	4
FALA rapido	175.8	160	16015	20 min	3	120.9	116	41177	34 min	0

Foi definido que o valor de custo mínimo do diagrama de montagem é o menor valor encontrado em todos os aprendizados testados. Percebe-se, que na estrutura de prolongamento horizontal as duas configurações testadas e as duas abordagens usadas, alcançaram o mesmo valor de custo mínimo. Nesse caso o melhor resultado foi o algoritmo PLA de configuração rápida, pois conseguiu chegar no menor valor de custo 8 vezes em 10 aprendizados. No caso do algoritmo FALA, a configuração segura obteve melhores resultados tanto em questão de tempo quanto no custo médio.

Para a estrutura de prolongamento vertical, o algoritmo FALA com configuração rápida não conseguiu encontrar o mesmo valor de custo das outras configurações e abordagens. Nessa estrutura as configurações seguras tiveram melhores resultados que as configurações rápidas.

6.1.4 Plano de Execução usando o PLA

Para a análise do plano de execução foram realizadas três simulações na estrutura do tipo cisterna usando memórias $J = 10; J = 50; J = 100$ e taxas de aprendizado iguais a λ_1=0.08; λ_1=0.2; λ_1=0.6. Os restantes dos parâmetros do PLA são os seguintes: K=1, L=10, n=1 e s_i=0.1.

A Tabela 8 mostra os valores encontrados a partir das simulações. Os resultados são a média do valor mínimo encontrado e o número de iterações para achar o mínimo valor.

Tabela 8 – Plano de execução com valores médios para cada parâmetro J e λ_1.

J	λ_1	Iteração média	Custo médio
10	0.08	3924	9218.9
50	0.08	3872.3	9333.5
100	0.08	3706.7	9325.7
10	0.2	3686.3	9156.3
50	0.2	3453.7	9116
100	0.2	3607.7	9048.5
10	0.6	3893.3	9072
50	0.6	3738	9080.5
100	0.6	2646.7	9117.8

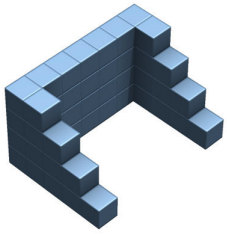
Percebe-se que altos valores de taxa de aprendizado (λ_1) tiveram resultados de custo médio menores que usando pequenos valores para taxa de aprendizado. Os valores de J maiores fizeram o algoritmo atingir o custo mínimo com um número de iterações menor do que quando utiliza-se valores de J pequenos.

Os melhores valores encontrados para os parâmetros são: $J = 50$ com $\lambda_1 = 0.2$ e $J = 100$ com $\lambda_1 = 0.2$. O primeiro chegou em um resultado mínimo mais rápido enquanto o segundo conseguiu alcançar o menor valor encontrado em relação a todos os testados.

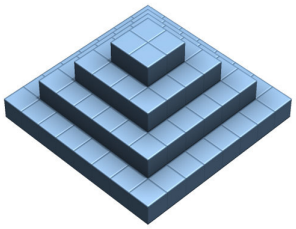
6.2 MONTAGEM DAS ESTRUTURAS EM AMBIENTE SIMULADO

Nesta seção é demonstrado o desempenho do diagrama de montagem e do plano de execução aprendidos. O V-Rep foi utilizado para realizar a montagem das estruturas no ambiente simulado. As estruturas propostas montadas no no software MagicalVoxel são ilustradas na Figura 47. Diferentemente dos blocos reais, os blocos do V-Rep não possuem a característica de autoalinhamento, o que pode causar erros de encaixe durante a montagem.

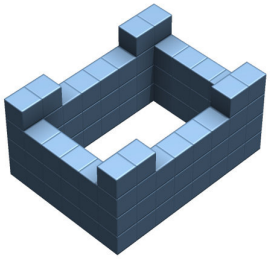
Figura 47 – Representação de estruturas no software MagicalVoxel.



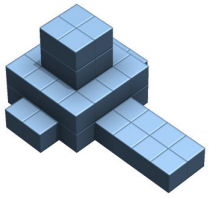
(a) Barragem



(b) Pirâmide



(c) Torre



(d) Estação Espacial

Para geração do diagrama de montagem das estruturas propostas são usados os seguintes parâmetros:

- PLA: $K = 1$, $L = 10$, $n = 1$, $\lambda_1 = 0.2$, $s_i = 0.1$ e $J = 100$
- FALA: $\lambda_1 = 0.2$ e $J = 100$

Para a geração do diagrama de montagem foram realizados 10 treinamentos para os algoritmos PLA e FALA, sendo que o resultado final é a média do custo mínimo encontrado. A geração do diagrama foi executada até um máximo de 5000 iterações.

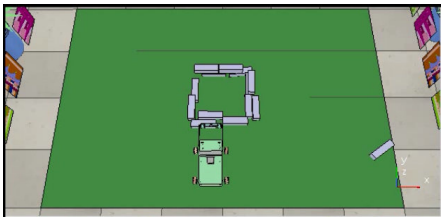
Para o plano de execução foram executados cinco treinamentos usando o diagrama de montagem da estrutura aprendida. Observe que somente o melhor plano de execução é usado na realização da tarefa de montagem. O número de iterações máximo utilizado é 4000 e os parâmetros utilizados são os seguintes: PLA: $K = 1$, $L = 10$, $n = 1$, $\lambda_1 = 0.2$, $s_i = 0.1$ e $J = 100$.

Para verificar o erro máximo de posição que ainda seja possível construir uma estrutura no ambiente simulado, foram realizadas três simulações da montagem da estrutura da cisterna. O valor da posição global é adquirido pelo V-Rep e junto com esse valor é inserido um ruído para ser enviado ao robô. A primeira simulação foi realizada com um valor de ruído que gere um erro de posicionamento igual a +/-3 cm. Durante a construção percebe-se que o robô tem dificuldades em centralizar o eletroímã para capturar o bloco da área de manipulação. A captura errada do bloco na área de manipulação e os erros de posicionamento acumulados durante o deslocamento do robô geram falhas de montagem.

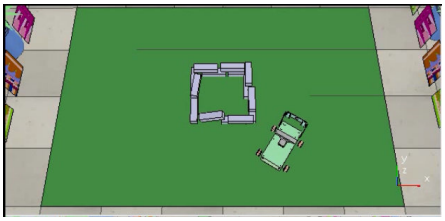
Em outra simulação o ruído gerou um erro de posicionamento de +/-2 cm e a situação foi semelhante com a simulação de erro de +/-3 cm. A diferença que existiu menos colisões do robô com os blocos e poucos blocos foram colocados nas suas posições corretas.

Com o teste de um ruído que gere erros de posicionamento de +/-0.8 cm, o robô conseguiu realizar corretamente a montagem da estrutura. Então estruturas podem ser corretamente construídas se tiver erros de posicionamento menores que +/-0.8 cm. A Figura 48 ilustra os três casos analisados em simulação para a construção de uma cisterna.

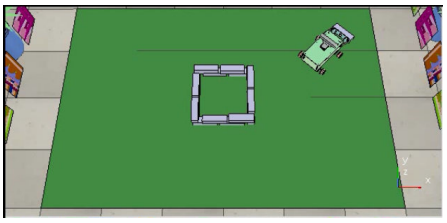
Figura 48 – Teste de montagem da estrutura com diferentes erros de posição.



(a) Erro de +/-3 cm



(b) Erro de +/-2 cm

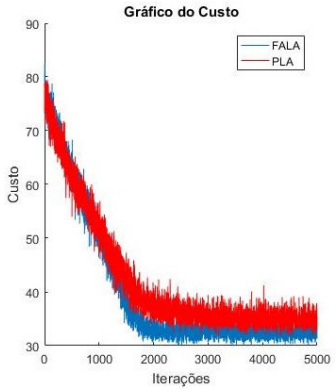


(c) Erro de +/-0.8 cm

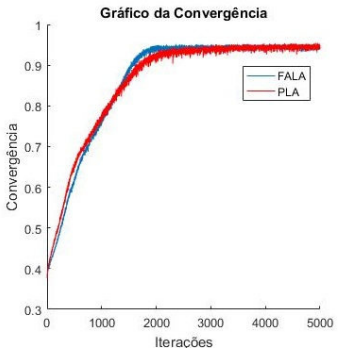
6.2.1 Estrutura do tipo Torre

A primeira estrutura construída foi a estrutura do tipo Torre. Essa estrutura possui somente uma camada de montagem com 4 níveis de altura. Foi gerado o diagrama de montagem com os algoritmos PLA e FALA. Para cada algoritmo foram executadas 10 simulações e feito a média dos valores de custo e convergência nas 5000 iterações. A Figura 49a mostra o comparativo do gráfico de custo médio e a Figura 49b mostra o comparativo do gráfico da convergência média.

Figura 49 – Gráfico do custo médio e da convergência média na geração do diagrama de montagem da estrutura tipo Torre.



(a) Custo médio



(b) Convergencia média

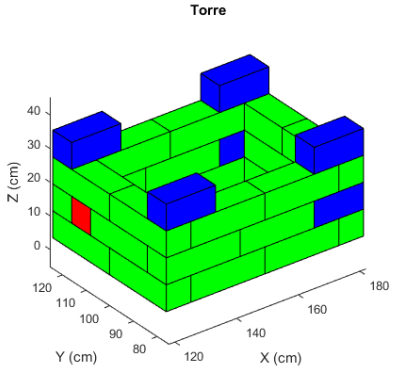
Percebe-se que os dois algoritmos apresentam uma curva de custo muito semelhantes, com a diferença que o algoritmo FALA possui resultados mais próximos do custo mínimo. Quanto a convergência, os dois métodos chegaram próximos de 1, com a diferença que o algoritmo FALA teve uma subida na convergência um pouco mais acentuada que o algoritmo PLA.

A Tabela 9 apresenta a média do menor custo, o custo mínimo encontrado, a média dos 100 últimos valores da convergência média e o número de blocos necessários para a construção da estrutura. A representação da estrutura de custo mínimo obtida a partir do diagrama de montagem é ilustrada na Figura 50.

Tabela 9 – Tabela de resultados do diagrama de montagem da estrutura do tipo Torre.

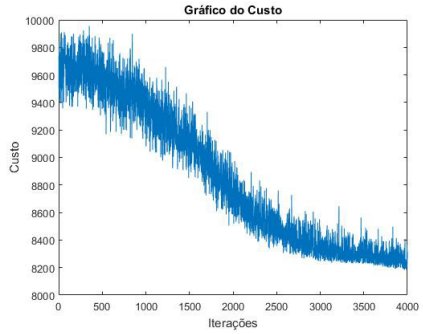
				Blocos		
Algoritmo	Custo médio	Custo mínimo	Convergência	1 unidade	2 unidade	3 unidade
FALA	30	30	0.9532	2	6	22
PLA	30	30	0.9542	2	6	22

Figura 50 – Representação da estrutura do tipo Torre gerada pelo diagrama de montagem aprendido.

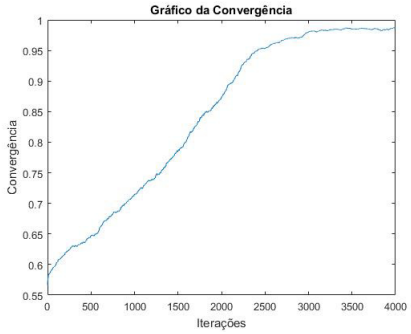


Note que esta é apenas uma representação da estrutura que o robô deverá aprender a construir. Ao finalizar o aprendizado do diagrama de montagem, o plano de execução é gerado. A Figura 51a apresenta o custo médio do plano de execução e a Figura 51b a convergência média.

Figura 51 – Gráfico do custo médio e da convergência média da estrutura do tipo Torre.



(a) Gráfico do erro



(b) Estrutura Montada

O plano de execução de menor custo é enviado para o robô para que ele execute os procedimentos de manipulação e de montagem. A sua velocidade linear foi configurada para um máximo de 0.03 m/s e a velocidade de rotação igual 0.08 rad/s. O robô demorou cerca de 84 minutos para realizar a montagem da estrutura, percorrendo uma distância total de 84.5 metros. A Tabela 10 mostra os resultados do plano ótimo de execução.

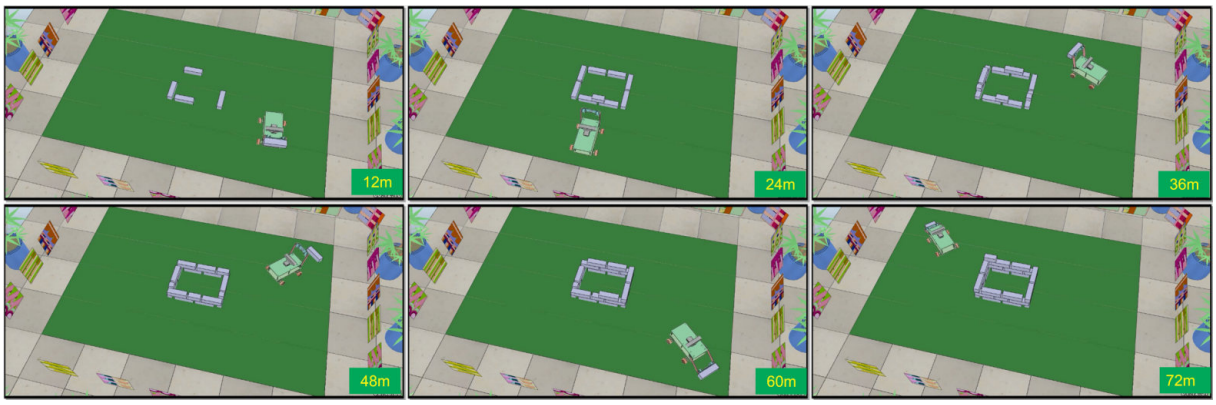
Tabela 10 – Resultados dos tempos e distâncias percorridas pelo robô na execução da tarefa de montagem da estrutura do tipo Torre.

	Valores
Tempo de Manipulação	36 min
Tempo de Montagem	48 min
Total	84 min

	Valores
Distância de Manipulação	38.4 m
Distância de Montagem	46.1 m
Total	84.5 m

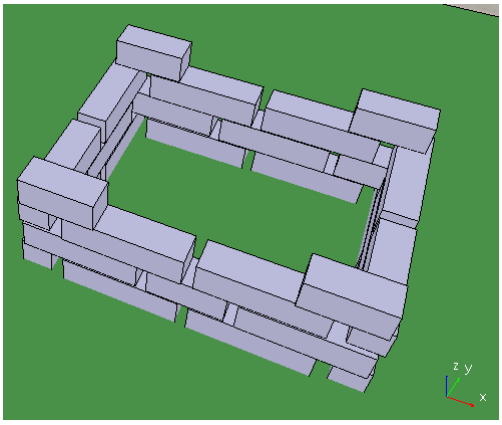
A Figura 52 ilustra a montagem da estrutura no software V-rep com a informação do tempo (em minutos) em que o quadro foi capturado.

Figura 52 – Resultados dos tempos e distâncias percorridas pelo robô na execução da tarefa de montagem da estrutura do tipo Torre.



A Figura 53 mostra a estrutura final montada.

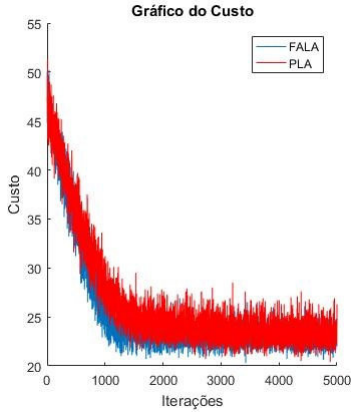
Figura 53 – Estrutura do tipo Torre montada.



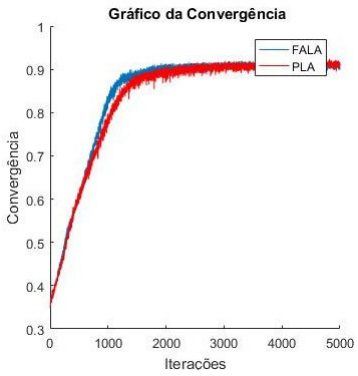
6.2.2 Estrutura do tipo Barragem

A segunda estrutura construída é a estrutura do tipo Barragem. Essa estrutura possui somente uma camada de montagem com 4 níveis de altura, semelhante a Torre, mas com um número menor de blocos. A Figura 54a mostra o comparativo do gráfico de custo do diagrama de montagem usando os algoritmos FALA e PLA. A Figura 54b mostra o comparativo do gráfico da convergência usando os dois algoritmos.

Figura 54 – Gráfico do custo médio e da convergência média na geração do diagrama de montagem da estrutura tipo Barragem.



(a) Custo médio.



(b) Convergência média

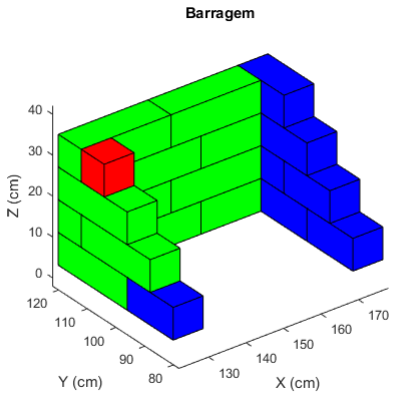
Percebe-se que os dois algoritmos apresentam uma curva de custo muito semelhantes, com a diferença que o algoritmo FALA chegou no custo mínimo mais rápido que o algoritmo PLA. Quanto a convergência, os dois métodos chegaram próximos de um, com a diferença que o algoritmo FALA teve uma subida na convergência um pouco mais acentuada que o algoritmo PLA.

A Tabela 11 apresenta a média do menor custo, o custo mínimo encontrado, a média dos 100 últimos valores da convergência média e o número de blocos necessários para a construção da estrutura. A representação da estrutura de custo mínimo obtida a partir do diagrama de montagem é ilustrada na Figura 55

Tabela 11 – Tabela de resultados da geração do diagrama de montagem da estrutura do tipo Barragem.

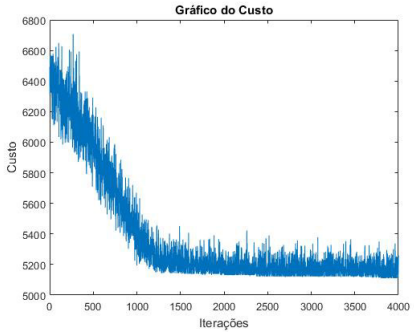
				Blocos		
Algoritmo	Custo médio	Custo mínimo	Convergência	1 unidade	2 unidades	3 unidades
FALA	19.2	19	0.9183	1	7	11
PLA	19.2	19	0.9210	1	7	11

Figura 55 – Representação da estrutura do tipo Barragem gerada pelo diagrama de montagem aprendido.

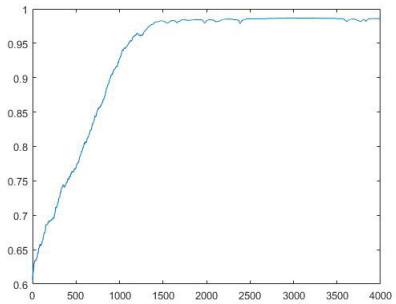


Ao finalizar o aprendizado do diagrama de montagem, o plano de execução é gerado. A Figura 56a apresenta o custo médio do plano de execução e a Figura 56b a convergência média.

Figura 56 – Gráfico do custo médio e da convergência média da estrutura do tipo Barragem.



(a) Gráfico do erro



(b) Estrutura Montada

O plano de execução de menor custo é enviado para o robô para que ele execute os procedimentos de manipulação e de montagem. A sua velocidade linear foi configurada para um máximo de 0.03 m/s e a velocidade de rotação igual 0.08 rad/s. O robô demorou cerca de 57 minutos para realizar a montagem da estrutura, percorrendo uma distância total de 58.2 metros. A Tabela 12 mostra os resultados do plano ótimo de execução.

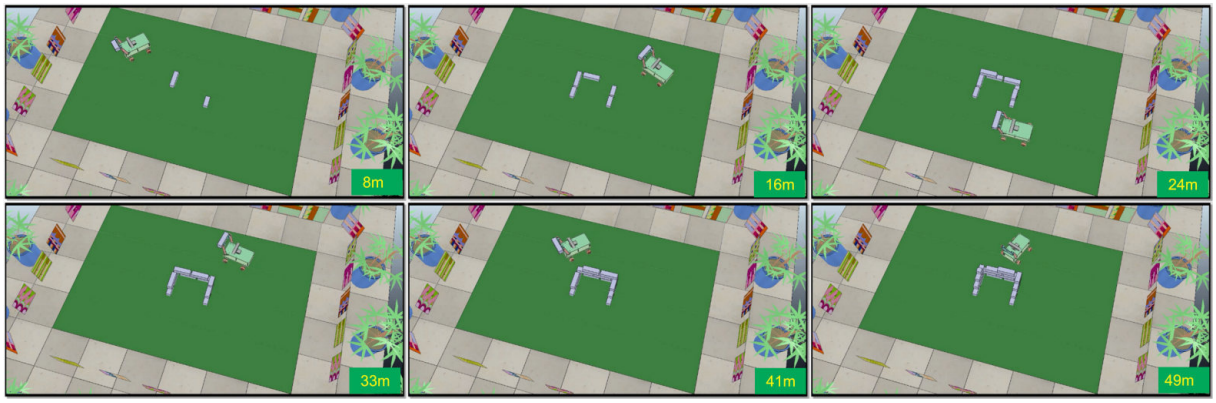
Tabela 12 – Resultados dos tempos e distâncias percorridas pelo robô na execução da tarefa de montagem da estrutura do tipo Barragem.

	Valores
Tempo de Manipulação	21 min
Tempo de Montagem	36 min
Total	57 min

	Valores
Distância de Manipulação	19.9 m
Distância de Montagem	38.3 m
Total	58.2 m

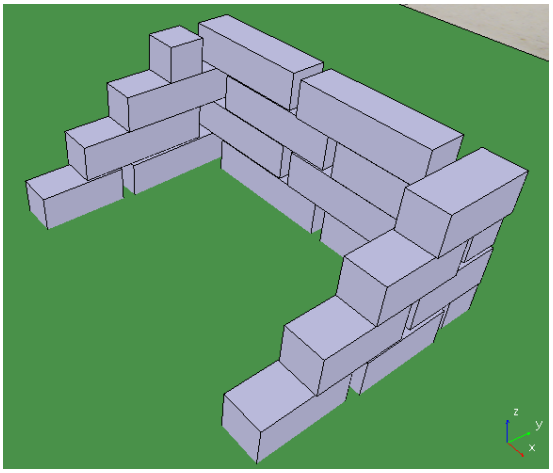
A Figura 57 ilustra a montagem da estrutura no software V-rep com a informação do tempo (em minutos) em que o quadro foi capturado.

Figura 57 – Montagem da estrutura do tipo Barragem.



A Figura 58 mostra a estrutura final montada.

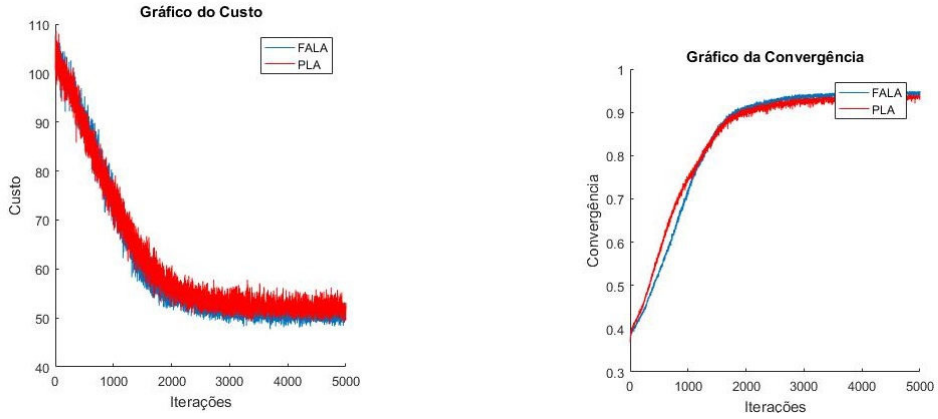
Figura 58 – Estrutura do tipo Barragem montada.



6.2.3 Estrutura do tipo Pirâmide

A terceira estrutura construída foi do tipo Pirâmide. Essa estrutura possui quatro camadas concêntricas com quatro níveis de altura. Essa é a maior estrutura em número de bloco construída pelo robô. A Figura 59a mostra o comparativo do gráfico de custo médio e a Figura 59b mostra o comparativo do gráfico da convergência média.

Figura 59 – Gráfico do custo médio e da convergência média na geração do diagrama de montagem da estrutura tipo Pirâmide.



(a) Custo médio (b) Convergência média

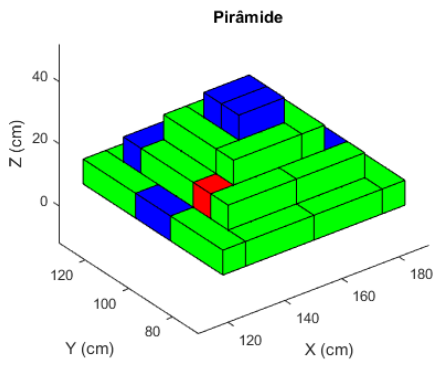
Percebe-se que os algoritmos PLA e o FALA apresentam uma curva de custo muito semelhantes, quase sem nenhuma diferença. Quanto a convergência, os dois métodos chegaram próximos de 1.

A Tabela 13 apresenta a média do menor custo, o custo mínimo encontrado, a média dos 100 últimos valores da convergência média e o número de blocos necessários para a construção da estrutura. A representação da estrutura de custo mínimo obtida a partir do diagrama de montagem é ilustrada na Figura 60

Tabela 13 – Tabela de resultados da geração do diagrama de montagem da estrutura do tipo Pirâmide.

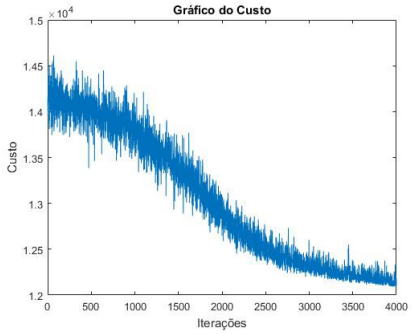
				Blocos		
Algoritmo	Custo médio	Custo mínimo	Convergência	1 unidade	2 unidades	3 unidades
FALA	46.2	46	0.9545	2	14	30
PLA	46.7	46	0.9453	2	14	30

Figura 60 – Representação da estrutura do tipo Pirâmide gerada pelo diagrama de montagem aprendido.

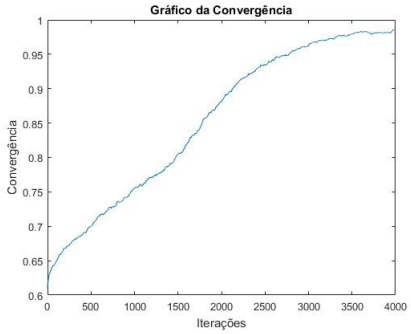


Ao finalizar o aprendizado do diagrama de montagem, o plano de execução é gerado. A Figura 61a apresenta o custo médio do plano de execução e a Figura 61b a convergência média.

Figura 61 – Gráfico do custo médio e da convergência média da estrutura do tipo Pirâmide.



(a) Gráfico do erro



(b) Estrutura Montada

O plano de execução de menor custo é enviado para o robô para que ele execute os procedimentos de manipulação e montagem. A sua velocidade linear foi configurada para um máximo de 0.03 m/s e a velocidade de rotação igual 0.08 rad/s. O robô demorou cerca de 120 minutos para realizar a montagem da estrutura, percorrendo uma distância total de 133.8 metros. A Tabela 14 mostra os resultados do plano ótimo de execução.

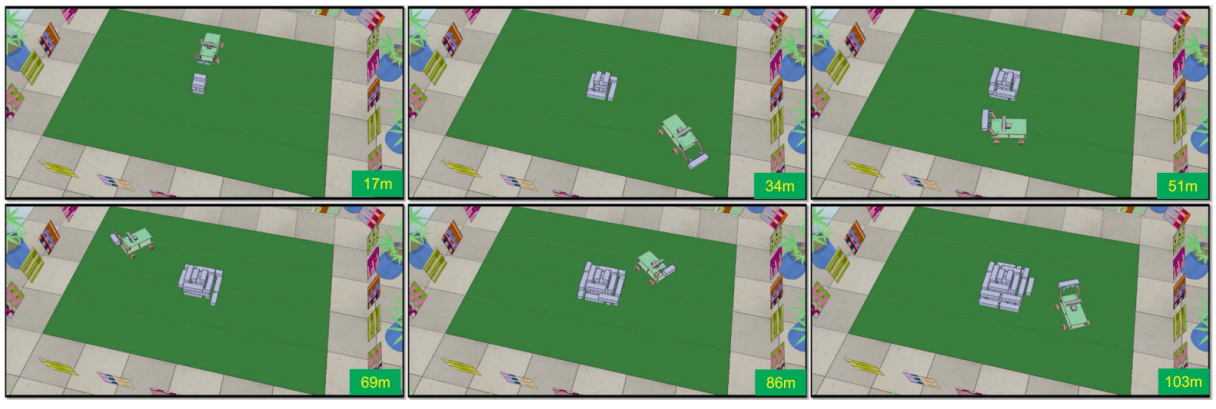
Tabela 14 – Resultados dos tempos e distâncias percorridas pelo robô na execução da tarefa de montagem da estrutura do tipo Pirâmide.

	Valores
Tempo de Manipulação	49 min
Tempo de Montagem	71 min
Total	120 min

	Valores
Distância de Manipulação	61.1 m
Distância de Montagem	72.7 m
Total	133.8 m

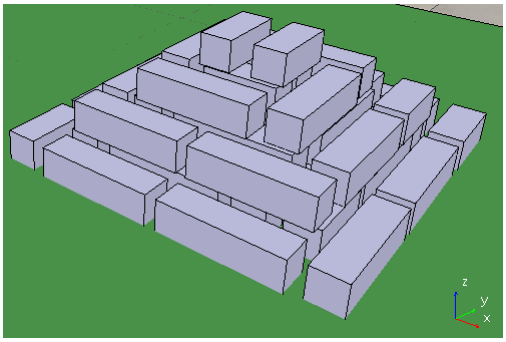
A Figura 62 ilustra a montagem da estrutura no software V-rep com a informação do tempo (em minutos) em que o quadro foi capturado.

Figura 62 – Montagem da estrutura do tipo Pirâmide.



A Figura 63 mostra a estrutura final montada.

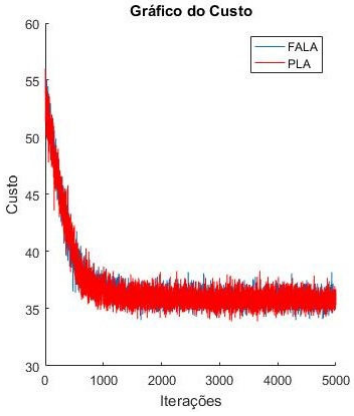
Figura 63 – Estrutura do tipo Pirâmide montada.



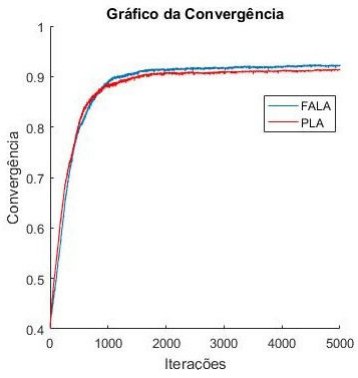
6.2.4 Estrutura do tipo Estação Espacial

A quarta estrutura construída foi a estrutura da Estação Espacial. Essa estrutura assim como a Pirâmide possui 4 camadas concêntricas com 4 níveis de altura, com a diferença que o número de blocos usados é bem menor. A Figura 64a mostra o comparativo do gráfico de custo da geração do diagrama de montagem usando o algoritmo FALA e o algoritmo PLA. A Figura 64b mostra o comparativo do gráfico da convergência usando os dois algoritmos.

Figura 64 – Gráfico do custo médio e da convergência média na geração do diagrama de montagem da Estação Espacial.



(a) Custo médio



(b) Convergência média

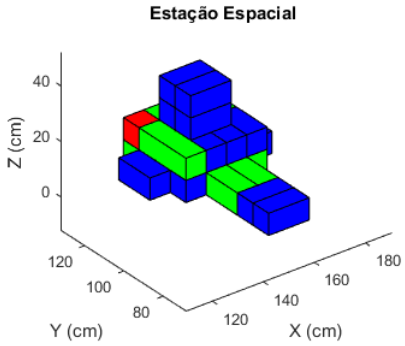
Percebe-se que os dois algoritmos apresentam uma curva de custo muito semelhantes. Quanto a convergência, os dois métodos chegaram próximos de 1, com a diferença que o algoritmo FALA teve resultados mais próximos de 1 que o algoritmo PLA.

A Tabela 15 apresenta a média do menor custo, o custo mínimo encontrado, a média dos 100 últimos valores da convergência média e o número de blocos necessários para a construção da estrutura. A representação da estrutura de custo mínimo obtida a partir do diagrama de montagem é ilustrada na Figura 65

Tabela 15 – Tabela de resultados do diagrama de montagem da estrutura do tipo Estação Espacial.

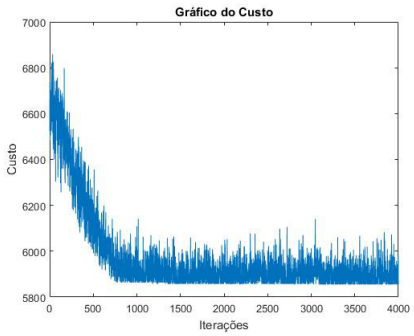
				Blocos		
Algoritmo	Custo médio	Custo mínimo	Convergência	1 unidade	2 unidades	3 unidades
FALA	32.9	32	0.9311	1	15	7
PLA	32.5	32	0.9232	1	15	7

Figura 65 – Representação da estrutura do tipo Estação Espacial gerada pelo diagrama de montagem aprendido.

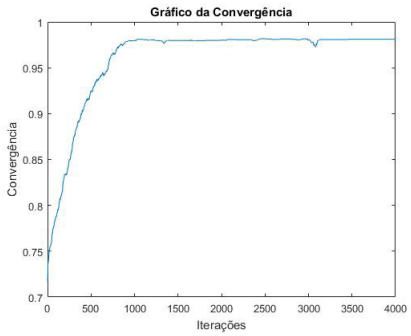


Ao finalizar o aprendizado do diagrama de montagem, o plano de execução é gerado. A Figura 66a apresenta o custo médio do plano de execução e a Figura 66b a convergência média.

Figura 66 – Gráfico do custo médio e da convergência média da estrutura do tipo Estação Espacial.



(a) Gráfico do erro



(b) Estrutura Montada

O plano de execução de menor custo é enviado para o robô para que ele execute os procedimentos de manipulação e montagem. A sua velocidade linear foi configurada para um máximo de 0.03 m/s e a velocidade de rotação igual 0.08 rad/s. O robô demorou cerca de 62 minutos para realizar a montagem da estrutura, percorrendo uma distância total de 65.3 metros. A Tabela 16 mostra os resultados do plano ótimo de execução.

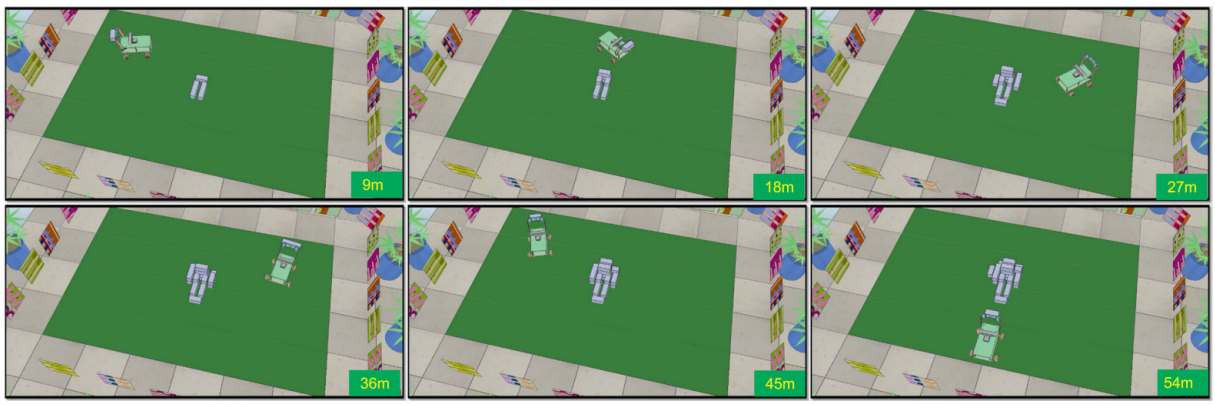
Tabela 16 – Resultados dos tempos e distâncias percorridas pelo robô na execução da tarefa de montagem da estrutura do tipo Estação Espacial.

	Valores
Tempo de Manipulação	26 min
Tempo de Montagem	36 min
Total	62 min

	Valores
Distância de Manipulação	30.1 m
Distância de Montagem	35.2 m
Total	65.3 m

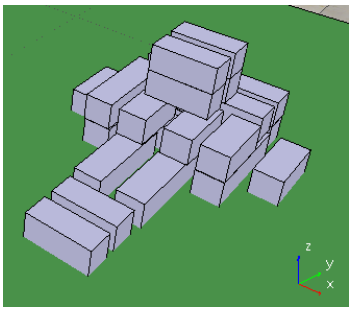
A Figura 67 ilustra a montagem da estrutura no software V-rep com a informação do tempo (em minutos) em que o quadro foi capturado.

Figura 67 – Montagem da estrutura do tipo Estação Espacial.



A Figura 68 mostra a estrutura final montada.

Figura 68 – Estrutura do tipo Estação Espacial montada.



6.2.5 Análise da montagem das estruturas

Em todas as montagens o erro médio de posicionamento foi menor que 1 mm. Dessa forma foi demonstrado que com baixo erro de posicionamento é possível realizar a montagem de estruturas com 4 níveis de altura e com várias camadas concêntricas.

Os gráficos de convergência de todos as abordagens usadas não chegaram ao valor unitário, esse fato é devido a função de avaliação de Eq. 5.6. Essa função necessita de baixos valores de custo para aumentar o valor de probabilidade. Quando o algoritmo atinge um valor

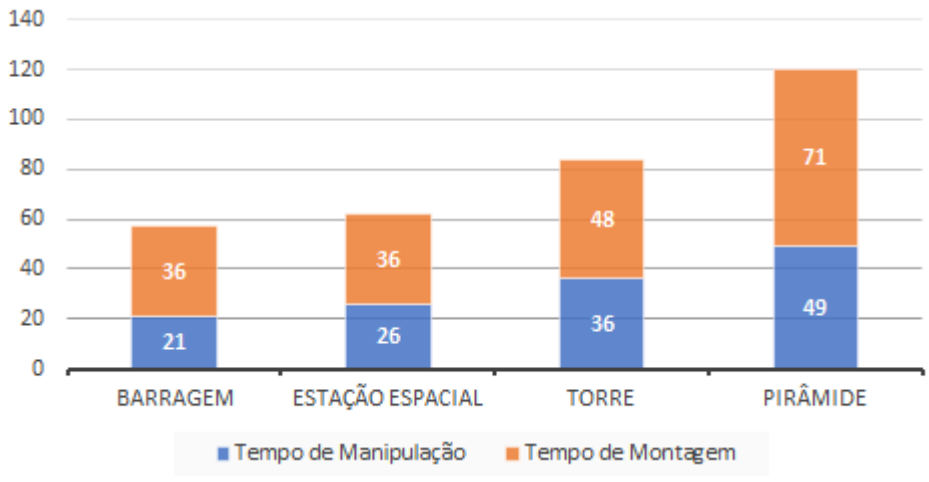
de custo mínimo, a função de avaliação retorna valores de recompensa com valores próximos à zero para os autômatos e, consequentemente, o gráfico de convergência não chega no valor unitário. Outro motivo é a existência de várias ações ótimas para alguns autômatos. Isso faz com que as ações tenham valores iguais de probabilidade. Então as probabilidades não chegam ao valor unitário e, consequentemente, o valor de convergência total não alcança o valor unitário.

Estruturas como por exemplo do tipo Pirâmide possui um número de camadas concêntricas maior que 1. Tornando uma estrutura difícil de ser montada pois a cada camada concêntricas montada, maior será a quantidade de obstáculos que o robô deve evitar.

Os diagramas de montagem das estruturas do tipo Barragem, Torre e Pirâmide, atingiram o menor custo com um valor igual ao número total de blocos usados, ou seja, todas as restrições quanto às falhas de conexão dos blocos foram superadas. A estrutura do tipo estação lunar não teve seu custo mínimo igual ao seu número de blocos, ou seja, o algoritmo de geração do diagrama de montagem teve que chegar em um valor de custo que diminuísse tanto o número de blocos quanto o número de falhas de conexão.

O gráfico da Figura 69 mostra os tempos de manipulação e montagem, em minutos, de cada uma das estruturas. Percebe-se que o tempo de montagem foi um pouco maior que o tempo de manipulação. Esse fato deve-se ao modo como o algoritmo do plano de execução fez a minimização do deslocamento total do robô. Esse fato pode ser motivo também da disposição das áreas de manipulação no ambiente.

Figura 69 – Tempos de Manipulação e de Montagem.

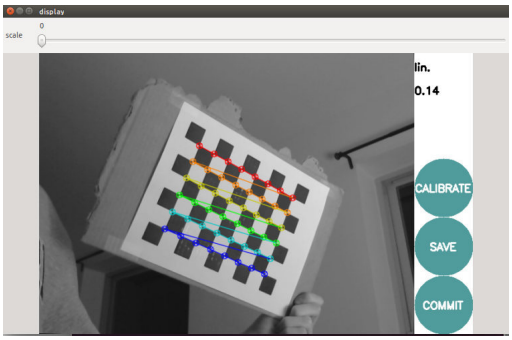


6.3 TESTES NO ROBÔ REAL

Para um trabalho que exige o uso de câmeras é importante que sejam tão próximas quanto possível do modelo ideal de projeção. Os dois principais motivos por causar distorções em câmeras reais são a distorção da lente e a descentralização.

Os procedimentos de calibração mais usuais são feitos com uma imagem plana com a imagem xadrez (Figura 70). Bibliotecas, como as que estão presentes no OpenCV, podem efetivamente modelar a distorção da lente e a descentralização. Após a calibração, o erro de uma câmera é aproximadamente de 0.1 a 0.2 pixels.

Figura 70 – Calibração feita em um aplicativo do ROS, usando uma imagem de xadrez.



O ROS fornece um pacote que permite a calibração de uma câmera monocular. Para realizar a calibração do dispositivo Asus Xtion PRO LIVE é preciso calibrar tanto a câmera RGB, quanto a câmera IR. Os parâmetros calibrados são os parâmetros intrínsecos, ou seja, eles são todos os parâmetros internos de uma câmera, tais como: distância focal, ponto principal, distorção das lentes, etc.

Os parâmetros intrínsecos adquiridos através da calibração das câmeras pelo ROS são mostrados na Tabela 17.

Tabela 17 – parâmetros intrínsecos do dispositivo Asus Xtion PRO LIVE.

Parâmetros	Câmera RGB	Câmera de profundidade
camera_matrix	$\begin{bmatrix} 543.065783 & 0 & 306.364771 \\ 0 & 545.341779 & 240.338943 \\ 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 580.235872 & 0 & 323.298253 \\ 0 & 574.736205 & 241.844413 \\ 0 & 0 & 1 \end{bmatrix}$
distortion_coefficients	$\begin{bmatrix} 0.064779 & -0.126981 & 0.002331 & -0.006970 & 0 \end{bmatrix}$	$\begin{bmatrix} -0.040436 & 0.015428 & -0.000859 & -0.000178 & 0 \end{bmatrix}$
projection_matrix	$\begin{bmatrix} 545.460999 & 0 & 301.618102 & 0 \\ 0 & 551.077515 & 240.783235 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$	$\begin{bmatrix} 569.539185 & 0 & 322.729296 & 0 \\ 0 & 564.220032 & 240.933111 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$

Diferentemente do robô simulado, o robô real possui erros de posicionamento durante seu deslocamento quando utiliza-se somente odometria. No cálculo de odometria é estimado uma posição global, mas devidos a escorregamentos e outros erros a posição global do robô é diferente. Para estimar o erro de posição no deslocamento do robô foram realizados testes com algumas trajetórias. No final da trajetória é verificado o erro da posição global do robô

em relação com a posição calculada da odometria. O primeiro teste foi feito fazendo o robô se deslocar 2 metros no eixo X 10 vezes seguidas. No segundo teste o robô realizou o deslocamento de 2 metros no eixo Y 10 vezes seguidas. No terceiro teste, o robô realizou vários movimentos rotacionais em torno do eixo Z 10 vezes seguidas. Os resultados são apresentados na Tabela 18.

Tabela 18 – Estimativa do erro no deslocamento do robô.

	Erro em X	Erro em Y	Erro em θ
Deslocamento em X	0.33 centímetros por metro	0.21 centímetros por metro	2º por metro
Deslocamento em Y	0.28 centímetros por metro	0.32 centímetros por metro	4º por metro
Rotação	0.03 centímetros por rotação	sem erro	7º por rotação

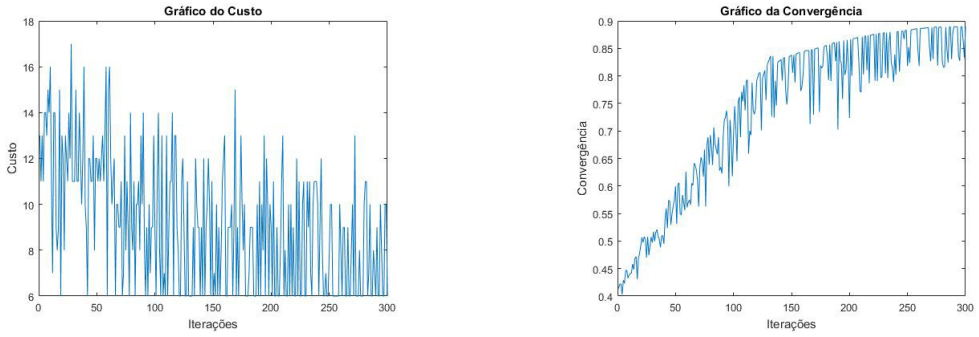
Percebe-se que o robô possui erros de posicionamento no seu deslocamento. Por esse motivo é usado o sistema de localização e mapeamento por visão. Dessa maneira o robô consegue diminuir o erro de posição acumulado durante seu deslocamento.

O ambiente usado na montagem possui alguns objetos próximos a parede para facilitar o reconhecimento de pontos de interesses no ambiente. O mapeamento é feito de forma não autônoma por meio de comandos via ROS.

Para a validação experimental do sistema, o robô real irá montar uma estrutura simplificada do tipo Parede. A estrutura do tipo parede possui 2 níveis de altura e 1 camada concêntrica. A geração da matriz de entrada M_{ent} é feita a partir da estrutura criada no MagicalVoxel, mostrada na Figura 72a. A partir da matriz de entrada M_{ent} é gerado o diagrama de montagem da estrutura. Os parâmetros usados no algoritmo de criação do diagrama são: PLA: $K = 1$, $L = 10$,$n = 1$, $\lambda_1 = 0.2$, $s_i = 0.1$ e $J = 100$.

Na Figura 71 são apresentados os gráficos de custo e de convergência.

Figura 71 – Gráfico do custo e da convergência na criação do diagrama de montagem.

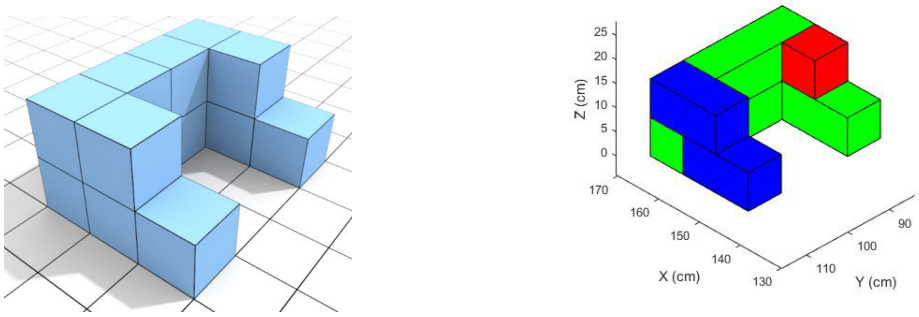


(a) Gráfico do custo

(b) Gráfico da convergência

O diagrama ótimo de montagem da estrutura da Figura 72a, usa 3 blocos de uma unidade, dois blocos de 2 unidades e 1 bloco de uma unidade. A estrutura gerada é mostrada na Figura 72b.

Figura 72 – Entrada do algoritmo de geração do diagrama de montagem e a sua saída de menor custo.



(a) Estrutura de entrada do algoritmo

(b) Estrutura de menor custo.

O ambiente possui somente uma área de manipulação. Por esse motivo e também devido ao baixo número de blocos, o plano de execução chegou no melhor resultado em poucas iterações. O resultado da ordem de montagem definido pelo plano de execução e a posição dos blocos na área de manipulação são vistos na Tabela 19.

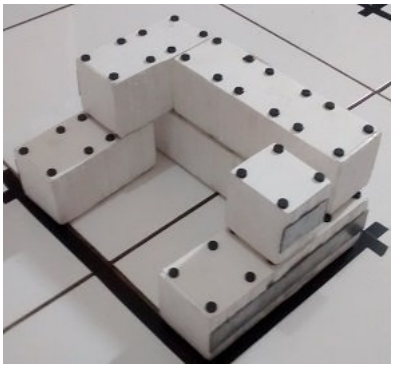
Tabela 19 – Plano de execução e a posição dos blocos na area de manipulação.

Ordem de Montagem					
Ordem	Posição X (cm)	Posição Y (cm)	Posição Z (cm)	Oritentação	Tipo de Bloco
1	146	112	4	0º	2
2	158	104	4	90º	3
3	150	88	4	0º	3
4	158	96	12	0º	3
5	154	112	12	90º	2
6	150	88	12	90º	1

Local de Manipulação				
Posição X (cm)	Posição Y (cm)	Posição Z (cm)	Oritentação	Tipo de Bloco
280	100	4	90º	1
280	100	4	90º	2
280	100	4	90º	3

Para montagem da estrutura são usados blocos de isopor com ímãs na sua superfície superior e inferior. A estrutura construída pelo robô real é mostrada 73.

Figura 73 – Estrutura montada.



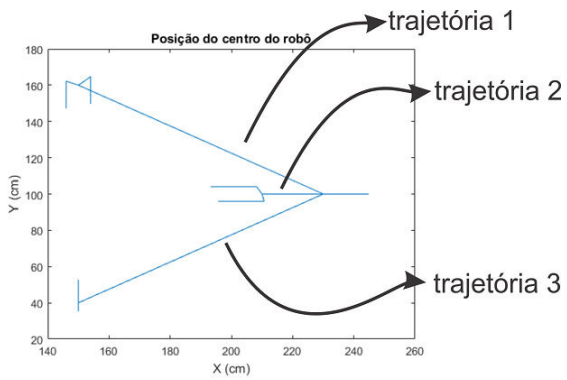
Para realização da montagem foi usado o RTAB-Map para o mapeamento e localização. Os parâmetros e outros relacionados ao ROS estão no Anexo A.

A primeira tentativa de montagem é feita com o RTAB-Map funcionando no modo de mapeamento e localização. O robô é posicionado em um ponto no ambiente a frente da área de manipulação, essa é a posição inicial do robô. O robô realiza a captura do bloco e leva até o ponto de destino, nesse momento o robô desloca-se usando somente os valores da odometria da roda para se localizar. Ao deixar o bloco no local de destino, o robô realiza seu trajeto de retorno para sua posição inicial. Chegando no ponto inicial, o algoritmo do RTAB-Map identifica que o local já foi visitado e realiza a correção da posição global do robô.

Para montagem do segundo bloco, o robô realiza sua trajetória em um caminho diferente do realizado para o procedimento anterior. O robô utiliza novamente os dados da odometria para se localizar.

A trajetória do robô para realizar a montagem da estrutura foi gerada pelo plano de execução e se resume em três trajetórias conforme ilustrado na Figura 74. A primeira vez que o robô realiza sua trajetória em algum dos caminhos são usados somente os valores de odometria para se localizar. Como demonstrado na Tabela 18, o robô possui erros de posição e orientação quando desloca-se, esses erros são corrigidos pelo RTAB-Map quando o robô desloca-se por um local já mapeado.

Figura 74 – Trajetorias geradas pelo plano de execução para realização da montagem.



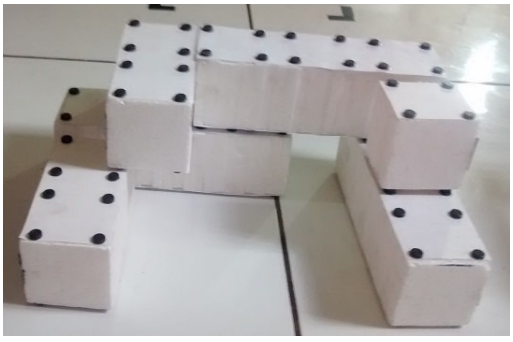
Para realização da montagem, todos os blocos devem ser posicionados com uma boa precisão. A primeira tentativa não foi bem sucedida pois foi preciso usar somente os dados da odometria em alguns momentos da montagem dos blocos. A Figura 75 mostra uma estrutura que foi montada enquanto o robô realizava o mapeamento e a localização.

Figura 75 – Estrutura montada enquanto o robô realiza o mapeamento e localização.

A segunda tentativa foi realizada com um mapeamento prévio do ambiente. Para o mapeamento do ambiente foram realizadas algumas trajetórias não autônomas. Ao terminar o mapeamento é iniciado o modo de localização do RTAB-Map e o robô já pode realizar a montagem da estrutura.

Com erros de posição menores que a primeira tentativa o robô já conseguiu realizar algumas montagens. Como existem erros quanto a posição global do robô, pois o algoritmo do RTAB-Map precisa dos valores de odometria para uma estimativa inicial de posição, as estruturas não foram montadas corretamente. Duas estruturas montadas são vistas na Figura 76.

Figura 76 – Estruturas montadas pelo robô considerando o erro de posição.



(a)

(b)

Na terceira tentativa, o mapeamento foi feito com as posições globais corretas de alguns pontos próximos da área de construção. Esse processo foi feito da seguinte maneira: 1) É feito o mapeamento do robô no ponto inicial; 2) O algoritmo do RTAB-Map é parado; 3) O robô

realiza o percurso até uma posição próxima da área de manipulação; 4) É Feito a correção do erro de posição do robô manualmente, deslocando o robô para a sua posição global correta; 5) O algoritmo do RTAB-Map é iniciado; 6) Repete desde a etapa dois até a cinco para o número de posições que desejar. Agindo dessa forma a posição global do robô é igual à posição calculada da odometria.

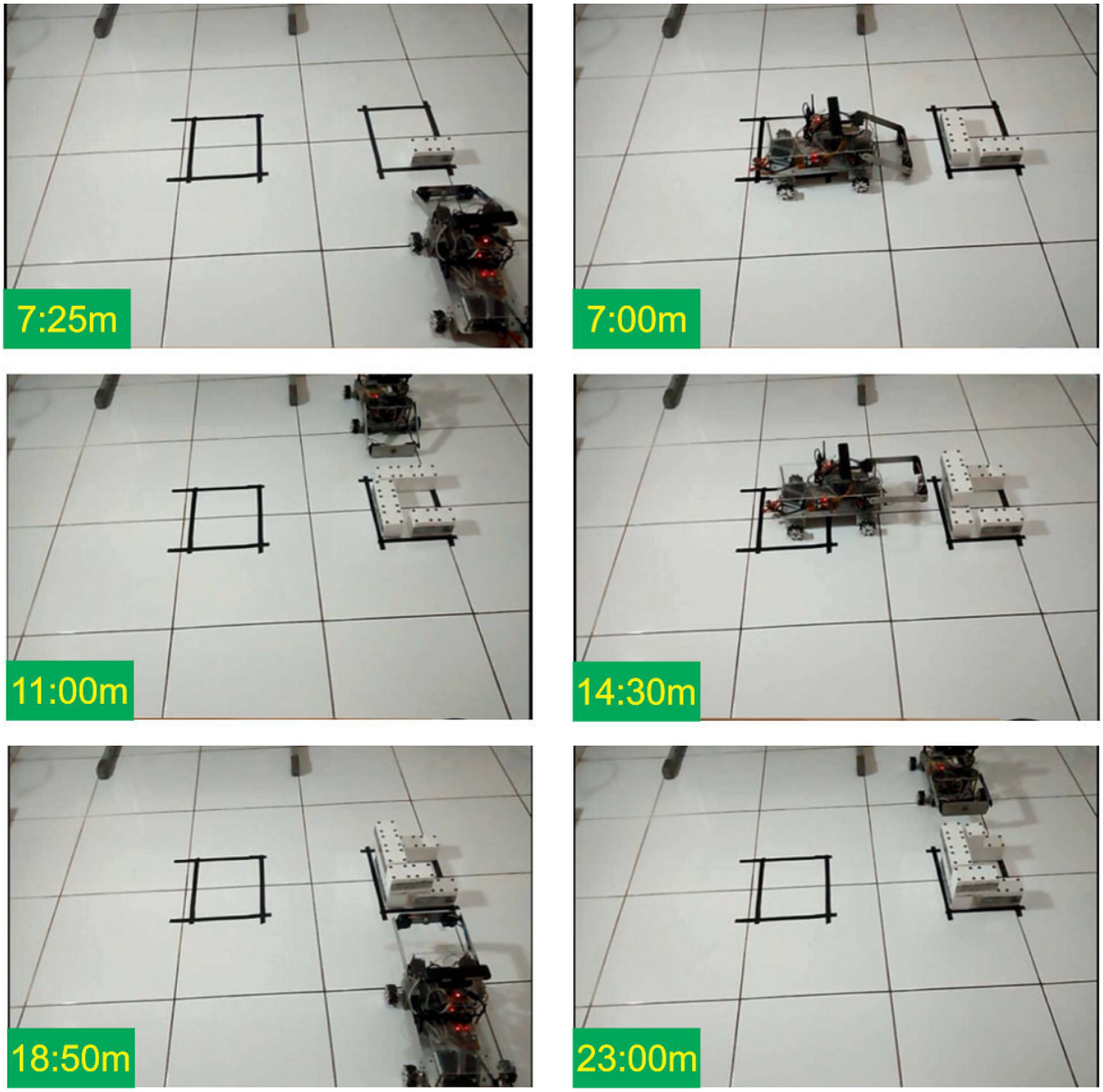
Foi feito o uso das etapas mencionadas para seis pontos próximos da área de montagem e a posição inicial. Assim, quando o robô tiver erros acumulados de posição e visitar algum ponto da área de montagem ou a posição inicial, a sua posição global é corrigida. A Figura 77 ilustra a representação 3D do ambiente a partir do mapa gerado pelo RTAB-Map.

Figura 77 – Mapa 3D gerado pelo RTAB-Map através do dispositivo Asus Xtion PRO LIVE.

Com a criação de um mapa com posições corretas, foi possível realizar o processo de montagem da estrutura. O processo de construção é ilustrado com imagens capturadas no momento que o robô deixa o bloco na área de montagem, esse processo é visto na Figura 78.

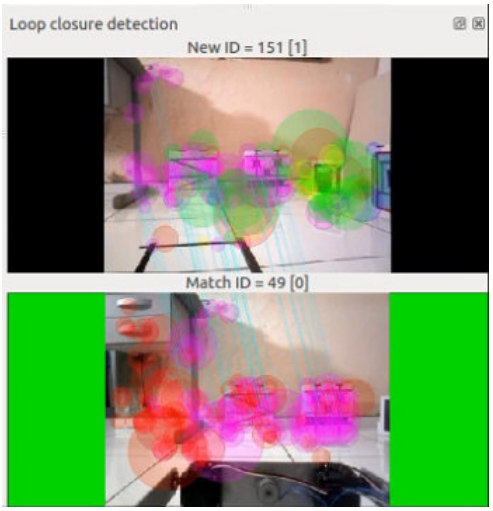
Dessa maneira a estrutura conseguiu ser montada com erros de posição muito baixos. A velocidade de deslocamento do robô foi ajustada no controlador *eband local planner* para uma velocidade máxima de 0.01m/s. Em baixas velocidades o processo de montagem foram melhores que em velocidades mais altas.

Figura 78 – Montagem da estrutura.

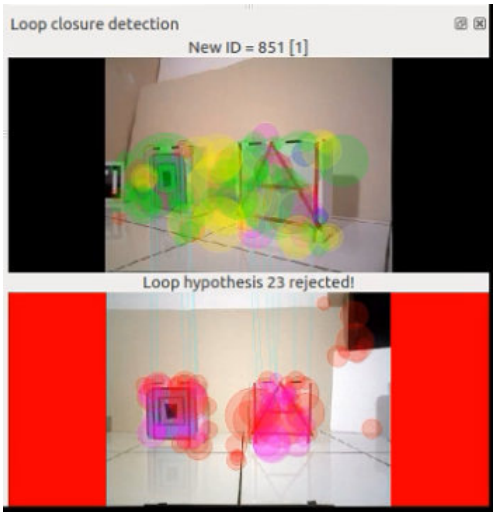


Um parâmetro do algoritmo do RTAB-Map de grande importância no reconhecimento de locais já visitados é o número de *inliers*. Esse parâmetro se for baixo, acaba encontrando falsos positivos e gerando uma estimativa de posição global errada. Se o valor for muito alto o algoritmo pode não reconhecer nenhuma posição já visitada. O valor de 30 *inliers*, foi definido para este trabalho. A Figura 79a mostra uma situação em que a posição foi aceita como visitada pois atingiu o critério de mais de 30 *inliers* e teve a sua posição global corrigida. A Figura 79b mostra uma situação em que é rejeitada a hipotese de se ter um fechameto de loop, por não ter o número de *inliers* suficientes.

Figura 79 – Aprovação ou rejeição de um fechamento de *loop* durante a montagem da estrutura.



(a) Detecção de um fechamento de *loop* durante a montagem da estrutura.

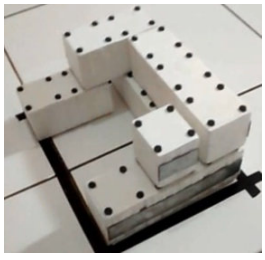


(b) Hipótese rejeitada de fechamento de *loop*.

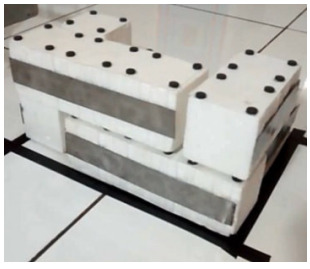
As imagens enviadas do Raspberry Pi para estação de controle foram definidas a uma taxa de duas imagens por segundo. Dessa maneira o Raspberry Pi não sofre de uma carga excessiva de processamento ao processar, compactar e enviar as imagens vindas do Asus Xtion PRO LIVE. O algoritmo do RTAB-Map envia estimativas de posição global para o controlador a cada um segundo.

A Figura 80c mostra outras montagens feitas pelo robô da mesma estrutura. O tempo médio da montagem das estruturas foram de aproximadamente 23 minutos. O erro médio de posicionamento do bloco é de aproximadamente 1 cm.

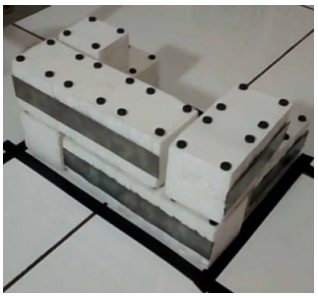
Figura 80 – Estruturas montadas com o uso de um mapa gerado a partir de posições conhecidas.



(a)



(b)



(c)

O autoalinhamento feito pelos ímãs, contribuiu para o sucesso da montagem da estrutura real, pois em situações em que o bloco estava um pouco afastado, o autoalinhamento corrige esse afastamento. Outro fator que contribuiu para o sucesso da montagem foi o uso das rodas mecanum (ou rodas suécas), permitindo pequenos ajustes de posição sem que o robô saia da área de montagem.

7 CONCLUSÃO

A construção de estruturas é um trabalho perigoso e que requer atenção nas etapas de planejamento e execução. O uso de robôs para executar a tarefa de construção ajuda na dinâmica das obras. Neste trabalho é realizado um estudo para estruturas de pequeno porte, porém a mesma estratégia pode ser aplicada em estruturas maiores.

Neste trabalho foi realizada a montagem de estruturas tridimensionais usando blocos de diferentes tamanhos. As gerações dos diagramas de montagem e do plano de execução foram obtidos através do método de RL denominado de *Learning Automata*. Para a realização da tarefa, um robô com a capacidade de locomover-se no ambiente (plano XY) foi construído para montar os blocos.

A metodologia apresentada neste trabalho está dividida em duas etapas: 1) a geração do diagrama de montagem; 2) a geração do plano de execução. A primeira etapa foi proposta para gerar o diagrama de montagem, onde os autômatos são armazenados em uma matriz em que cada ação desses autômatos representa um tipo de bloco. A seleção da ação em um determinado passo da iteração influência na escolha do próximo autômato da estrutura de planejamento.

Na primeira etapa foi abordado o uso de dois algoritmos, o PLA e o FALA. Ambas abordagens forneceram bons resultados em simulação. Porém o PLA achou os melhores resultados com maior frequência que o FALA. Este por sua vez, em algumas simulações, ficou preso em mínimos locais, enquanto o PLA em algumas situações conseguiu sair desses mínimos locais depois de algumas iterações.

A segunda etapa consiste em gerar o plano de execução. A arquitetura de planejamento é formada por uma rede de autômatos, onde alguns autômatos são responsáveis por selecionar as posições na área de manipulação e alguns outros autômatos são responsáveis por selecionar posições na área de montagem. O processo começa com um autômato de manipulação e, em seguida, um autômato de montagem. Esse processo repete-se até que a estrutura tenha sido totalmente concluída. Nesta etapa foi usado o PLA para realizar o processo de aprendizagem do plano de execução.

O algoritmo da segunda etapa conseguiu bons resultados, quando os procedimentos de manipulação e de montagem são obtidos a partir da combinação entre o PLA e o A*. No entanto, o tempo gasto para encontrar a solução é aumentada em ambientes muito grandes.

A adequada seleção dos melhores parâmetros para os algoritmos de geração do diagrama de montagem e do o plano de execução garantiu bons resultados nas quatro estruturas propostas. Os melhores parâmetros para o diagrama de montagem foram: 1) PLA: $K = 1$, $L = 10$, $n = 1$, $\lambda_1 = 0.2$, $s_i = 0.1$ e $J = 100$; 2) FALA: $\lambda_1 = 0.2$ e $J = 100$. Por outro lado, para o plano de execução foi: 3) PLA: $K = 1$, $L = 10$, $n = 1$, $\lambda_1 = 0.2$, $s_i = 0.1$ e $J = 100$. Em quase todos os testes realizados foram encontrados os mesmos valores de custo no diagrama de montagem e

no plano de execução, mostrando que os parâmetros usados são satisfatórios para encontrar a solução.

O plano de execução enviado para o robô simulado no V-Rep gerou boas trajetórias, pois o robô não colidiu em nenhum momento com os blocos da estrutura e conseguiu realizar todo o processo de montagem.

Durante a montagem da estrutura em simulação foi usada a posição global proveniente do V-Rep. O erro máximo obtido para realizar a montagem das peças foi de aproximadamente +/-1 mm. Esse resultado mostrou-se adequado já que para a estrutura não ser montada o erro teria que ser maior que +/-2 cm.

Os resultados da montagem simulada apresentaram o desempenho esperado. Pois para todos os casos testados o robô conseguiu montar a estrutura com o menor número de blocos e um menor tempo possível.

Para a montagem real, foram feitas calibrações nas câmeras do dispositivo Asus Xtion PRO LIVE para diminuir os possíveis erros produzidos pelas lentes. Percebeu-se que os dados de profundidade da câmera são adquiridos apenas em distâncias maiores ou igual a 65 cm do objeto, enquanto o fabricante informa que a distância mínima é de 80 cm.

O dispositivo Asus Xtion PRO LIVE possui uma não linearidade na informação de profundidade, ou seja, se ele for apontado para uma parede lisa, cada pixel da imagem fornece um valor de profundidade para mais ou para menos em relação ao valor real. Essa não linearidade aumenta em grandes distâncias.

O erro de estimativa de posição ocorre quando o robô realiza movimentos nos eixos X, Y e rotação no eixo Z. Essa estimativa foi obtida a partir de medições durante o deslocamento do robô em trajetórias lineares no eixo X e Y e rotações feitas no eixo Z. Percebeu-se que o robô apresenta erros que se acumulam durante o percurso. Esses erros fazem com que a estrutura não possa ser montada. Para solucionar esse problema foi usado um sistema de mapeamento e localização por visão, chamado de RTAB-Map.

A etapa de ajuste do algoritmo RTAB-Map para permitir o uso no processo de construção foi bastante árdua. Este algoritmo possui mais de 300 parâmetros e sua operação (para a criação de um bom mapa) pode ser bastante influenciada pela iluminação, número de objetos e predominância de cores. Percebeu-se que o ambiente deve ter uma boa iluminação artificial, pois além do uso da câmera RGB, o RTAB-Map usa imagens de profundidade.

Para garantir que exista variações de profundidade foram usados objetos distribuídos no ambiente. Dessa maneira, o RTAB-Map teria imagens de profundidade e de cor. O ambiente foi criado em um local pequeno para que a não linearidade da câmera de profundidade não prejudicasse o processo de mapeamento.

Para a comunicação ROS entre o Raspberry e a estação solo foi usado um roteador *wifi*

dedicado a essa função, pois somente a transferências das imagens consome uma banda de 6MB/s. Se o roteador for compartilhado com outras atividades, como por exemplo o acesso à internet ou a transferência de dados, o robô pode sofrer com a lentidão na transmissão ou no recebimento do seus dados, podendo causar até mesmo a perda de dados.

O robô real conseguiu realizar a montagem da estrutura proposta com poucas falhas de operações e pequeno erro de posição no momento da montagem. Assim pode-se concluir que o sistema de construção proposto neste trabalho conseguiu realizar com êxito todo processo de construção autônomo, tanto no ambiente simulado quanto no ambiente real, onde estão presentes vários fatores que prejudicam o processo de montagem.

7.1 INOVAÇÕES TECNOLÓGICAS OBTIDAS NESTE TRABALHO

Esse trabalho apresentou um sistema completo de construção autônomo, desde a criação do robô até o desenvolvimento dos algoritmos de aprendizagem do diagrama de montagem e do plano de execução. O processo de montagem foi realizado em um ambiente simulado com a montagem de 4 estruturas e no ambiente real com a montagem de uma estrutura.

Pode-se citar as seguintes inovações tecnológicas obtidas nesta pesquisa:

- Definição dos tipos de blocos e o melhor material para a realização da montagem. Além de propor uma maneira de realizar a captura dos blocos usando um eletroímã acoplado no braço do robô;
- Desenvolvimento de um robô móvel terrestre capaz de realizar o mapeamento e localização no ambiente. Esse robô foi desenvolvido para realizar a montagem de estruturas;
- Desenvolvimento do ambiente de simulação no V-Rrep para montagem de qualquer tipo de estrutura, desde que respeite as limitações do robô;
- Desenvolvimento dos algoritmos para geração do diagrama de montagem e do plano de execução usando um método de aprendizado por reforço denominado de *Learning Automata*;
- Desenvolvimento de códigos exclusivos para sistemas de montagem que são capazes de rodar em ambiente ROS. Esses códigos podem ser utilizados em robôs diferentes desde que mude alguns parâmetros.

7.2 TRABALHOS FUTUROS

Algumas sugestões para trabalhos futuros usando como base o trabalho já desenvolvido:

- Mudar o algoritmo para que ele possa ser usado por robôs aéreos, ou em ação conjunta com robôs terrestres;

- Aumentar o número de robôs construindo a estrutura ao mesmo tempo, em uma configuração de múltiplos robôs;
- Testar a abordagem proposta para estruturas que usem um elevado número de blocos;
- Usar um sistema de localização com precisão milimétrica;
- Usar uma estratégia para combinar a odometria visual com a odometria das rodas para obter melhores resultados na estimação da localização;
- Implementar um sistema de visão embarcada para aprimorar a precisão durante o processo de montagem das peças;
- Implementar sensores de força na garra para identificar a captura da peça;
- Implementar outros algoritmos de RL que sejam capazes de gerar tanto o plano de montagem quanto o plano de navegação, simultaneamente;
- Implementar algoritmos que sejam capazes de lidar com as falhas de montagem das peças;
- Investigar métodos de controle para lidar com as falhas dos atuadores do robô;
- Investigar outros algoritmos de inteligência artificial para a geração dos procedimentos de montagem;
- Investigar métodos de SLAM que possam ser aplicados ao sistema de construção e utilizem uma IMU(*Inertial Measurement Unit*) com um sensor auxiliar para a navegação;
- Implementar um algoritmo de navegação INS(*Inertial Navigation System*)/GPS(*Global Positioning System*) para operação em ambientes externos.

Referências

ARDINY, H.; WITWICKI, S. J.; MONDADA, F. Are Autonomous Mobile Robots Able to Take Over Construction? a Review. *International Journal of Robotics*, v. 4, n. 3, p. 10–21, 2015.

ASUS. *ASUS Xtion Pro Live*. 2017. Disponível em: <https://www.asus.com/us/3D-Sensor/Xtion_PRO_LIVE/>. Acesso em: 31 maio 2017.

AURELIO, D. do. *Amorfo no dicionário do aurélio de português.* 2017. Disponível em: <https://dicionariodoaurelio.com/>. Acesso em: 31 maio 2017.

CABRAL, K. M.; SANTOS, S. R. B. dos; GIVIGI, S. N.; NASCIMENTO, C. L. Design of model predictive control via learning automata for a single UAV load transportation. In: *2017 Annual IEEE International Systems Conference (SysCon)*. [S.l.]: IEEE, 2017.

CAI, Z.; HAN, J.; LIU, L.; SHAO, L. RGB-d datasets using microsoft kinect or similar sensors: a survey. *Multimedia Tools and Applications*, Springer Nature, v. 76, n. 3, p. 4313–4355, mar 2016.

CRUZ, L.; LUCIO, D.; VELHO, L. Kinect and RGBD images: Challenges and applications. In: *2012 25th SIBGRAPI Conference on Graphics, Patterns and Images Tutorials*. [S.l.]: IEEE, 2012.

DOBASHI, H.; HIRAOKA, J.; FUKAO, T.; YOKOKOHJI, Y.; NODA, A.; NAGANO, H.; NAGATANI, T.; OKUDA, H.; TANAKA, K. ichi. Robust grasping strategy for assembling parts in various shapes. *Advanced Robotics*, Informa UK Limited, v. 28, n. 15, p. 1005–1019, jun 2014.

DOGAR, M.; SPIELBERG, A.; BAKER, S.; RUS, D. Multi-robot grasp planning for sequential assembly operations. In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. [S.l.]: IEEE, 2015.

DORFLER, K. *Building strategies for on-site robotic construction*. 2017. Disponível em: <http://gramaziokohler.arch.ethz.ch/web/e/forschung/273.html>. Acesso em: 31 maio 2017.

ESMAEILPOUR, M.; NADERIFAR, V.; SHUKUR, Z. Design pattern mining using distributed learning automata and DNA sequence alignment. *PLoS ONE*, Public Library of Science (PLoS), v. 9, n. 9, p. e106313, sep 2014.

FOX, D.; BURGARD, W.; THRUN, S. The dynamic window approach to collision avoidance. *IEEE Robotics & Automation Magazine*, Institute of Electrical and Electronics Engineers (IEEE), v. 4, n. 1, p. 23–33, mar 1997.

GERSHENFELD, N.; CARNEY, M.; JENETT, B.; CALISCH, S.; WILSON, S. Macrofabrication with digital materials: Robotic assembly. *Architectural Design*, Wiley-Blackwell, v. 85, n. 5, p. 122–127, sep 2015.

HASHEM, M. K. *Learning Automata Based Intelligent Tutorial-like Systems*. Tese (Doutorado) — Carleton University, Ottawa, Ont., Canada, Canada, 2007. AAINR27099.

HELM, V.; ERCAN, S.; GRAMAZIO, F.; KOHLER, M. Mobile robotic fabrication on construction sites: DimRob. In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. [S.l.]: IEEE, 2012.

HERIK, H. J. van den; HENNES, D.; KAISERS, M.; TUYLS, K.; VERBEECK, K. Multi-agent learning dynamics: A survey. In: *Cooperative Information Agents XI*. [S.l.]: Springer Berlin Heidelberg, 2007. p. 36–56.

HOWELL, M. The application of continuous action reinforcement learning automata to adaptive PID tuning. In: *IEE Seminar Learning Systems for Control*. [S.l.]: IEE, 2000.

HOYT, R. P. SpiderFab: An architecture for self-fabricating space systems. In: *AIAA SPACE 2013 Conference and Exposition*. [S.l.]: American Institute of Aeronautics and Astronautics, 2013.

HUANG, A. S.; BACHRACH, A.; HENRY, P.; KRAININ, M.; MATURANA, D.; FOX, D.; ROY, N. Visual odometry and mapping for autonomous flight using an RGB-d camera. In: *Springer Tracts in Advanced Robotics*. [S.l.]: Springer International Publishing, 2016. p. 235–252.

IBRAHIM, I.; IBRAHIM, Z.; AHMAD, H.; YUSOF, Z. M. An assembly sequence planning approach with a multi-state particle swarm optimization. In: *Trends in Applied Knowledge-Based Systems and Data Science*. [S.l.]: Springer International Publishing, 2016. p. 841–852.

JIMÉNEZ, P. Survey on assembly sequencing: a combinatorial and geometrical perspective. *Journal of Intelligent Manufacturing*, Springer Nature, v. 24, n. 2, p. 235–250, aug 2013.

Jokic, S.; Novikov, P.; Maggs, S.; Sadan, D.; Jin, S.; Nan, C. Robotic positioning device for three-dimensional printing. *ArXiv e-prints*, jun. 2014.

KARSAI, I.; PÉNZES, Z. Comb building in social wasps: Self-organization and stigmergic script. *Journal of Theoretical Biology*, Elsevier BV, v. 161, n. 4, p. 505–525, apr 1993.

KHOSHELHAM, K.; ELBERINK, S. O. Accuracy and resolution of kinect depth data for indoor mapping applications. *Sensors*, MDPI AG, v. 12, n. 12, p. 1437–1454, feb 2012.

KHOSHNEVIS, B.; BODIFORD, M.; BURKS, K.; ETHRIDGE, E.; TUCKER, D.; KIM, W.; TOUTANJI, H.; FISKE, M. Lunar contour crafting - a novel technique for ISRU-based habitat development. In: *43rd AIAA Aerospace Sciences Meeting and Exhibit*. [S.l.]: American Institute of Aeronautics and Astronautics, 2005.

KINECT. *Microsoft Kinect*. 2017. Disponível em: <https://developer.microsoft.com/pt-br/windows/kinect>. Acesso em: 31 maio 2017.

KNEPPER, R. A.; LAYTON, T.; ROMANISHIN, J.; RUS, D. IkeaBot: An autonomous multi-robot coordinated furniture assembly system. In: *2013 IEEE International Conference on Robotics and Automation*. [S.l.]: IEEE, 2013.

LABBE, M.; MICHAUD, F. Appearance-based loop closure detection for online large-scale and long-term operation. *IEEE Transactions on Robotics*, Institute of Electrical and Electronics Engineers (IEEE), v. 29, n. 3, p. 734–745, jun 2013.

LABBE, M.; MICHAUD, F. Online global loop closure detection for large-scale multi-session graph-based SLAM. In: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. [S.l.]: IEEE, 2014.

LEE, H. W. Implementation of Localization System using Learning Automata based Sensor Fusion in Unmanned Forklifts. v. 11, n. 10, p. 6983–6989, 2016.

LINDRUP, V. S. *Robotic Maintenance and ROS Appearance Based SLAM and Navigation With a Mobile Robot Prototype*. 2016.

LINDSEY, Q.; MELLINGER, D.; KUMAR, V. Construction with quadrotor teams. *Autonomous Robots*, Springer Nature, v. 33, n. 3, p. 323–336, jun 2012.

MAGNENAT, S.; PHILIPPSEN, R.; MONDADA, F. Autonomous construction using scarce resources in unknown environments. *Autonomous Robots*, Springer Nature, v. 33, n. 4, p. 467–485, may 2012.

MARKOPOULOU, A. *Grip Robot: research group at the Institute for Advanced Architecture of Catalonia*. 2017. Disponível em: <http://robots.iaac.net/>. Acesso em: 31 maio 2017.

MCEVOY, M.; KOMENDERA, E.; CORRELL, N. Assembly path planning for stable robotic construction. In: *2014 IEEE International Conference on Technologies for Practical Robot Applications (TePRA)*. [S.l.]: IEEE, 2014.

MELLO, L. H. de; SANDERSON, A. A correct and complete algorithm for the generation of mechanical assembly sequences. *IEEE Transactions on Robotics and Automation*, Institute of Electrical and Electronics Engineers (IEEE), v. 7, n. 2, p. 228–240, apr 1991.

MISRA, S.; KRISHNA, P. V.; SARITHA, V.; OBAIDAT, M. S. Learning automata as a utility for power management in smart grids. *IEEE Communications Magazine*, Institute of Electrical and Electronics Engineers (IEEE), v. 51, n. 1, p. 98–104, jan 2013.

MOHAMMADI, M.; FRANCHI, A.; BARCELLI, D.; PRATTICHIZZO, D. Cooperative aerial tele-manipulation with haptic feedback. In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. [S.l.]: IEEE, 2016.

NAPP, N.; RAPPOLI, O. R.; WU, J. M.; NAGPAL, R. Materials and mechanisms for amorphous robotic construction. In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. [S.l.]: IEEE, 2012. p. 4879–4885.

NEWCOMBE, R. A.; DAVISON, A. J.; IZADI, S.; KOHLI, P.; HILLIGES, O.; SHOTTON, J.; MOLYNEAUX, D.; HODGES, S.; KIM, D.; FITZGIBBON, A. KinectFusion: Real-time dense surface mapping and tracking. In: *2011 10th IEEE International Symposium on Mixed and Augmented Reality*. [S.l.]: IEEE, 2011.

OCCIPITAL. *OCCIPITAL*. 2017. Disponível em: <https://structure.io/>. Acesso em: 31 maio 2017.

OLIVER, A.; KANG, S.; WüNSCHE, B. C.; MACDONALD, B. Using the kinect as a navigation sensor for mobile robotics. In: *Proceedings of the 27th Conference on Image and Vision Computing New Zealand - IVCNZ12*. [S.l.]: ACM Press, 2012.

PEDRAZA, G.; DIAZ, M.; LOMBERA, H. An approach for assembly sequence planning by genetic algorithms. *IEEE Latin America Transactions*, Institute of Electrical and Electronics Engineers (IEEE), v. 14, n. 5, p. 2066–2071, may 2016.

POUNDS, P. E. I.; BERSAK, D. R.; DOLLAR, A. M. Stability of small-scale UAV helicopters and quadrotors with added payload mass under PID control. *Autonomous Robots*, Springer Nature, v. 33, n. 1-2, p. 129–142, feb 2012.

QIOA, B.; TAHERI, H.; GHAEMINEZHAD, N. Kinematic Model of a Four Mecanum Wheeled Mobile Robot. *International Journal of Computer Applications*, v. 113, n. 3, p. 6–9, 2015.

QUINLAN, S.; KHATIB, O. Elastic bands: Connecting path planning and control. In: *In Proceedings of the International Conference on Robotics and Automation*. [S.l.: s.n.], 1993. p. 802–807.

ROESMANN, C.; FEITEN, W.; WOESCH, T.; HOFFMANN, F.; BERTRAM, T. Trajectory modification considering dynamic constraints of autonomous robots. In: *ROBOTIK 2012; 7th German Conference on Robotics*. [S.l.: s.n.], 2012. p. 1–6.

ROS. *ROS*. 2017. Disponível em: <http://www.ros.org/>. Acesso em: 31 maio 2017.

SAIDI, K. S.; O'BRIEN, J. B.; LYTLE, A. M. Robotics in construction. In: *Springer Handbook of Robotics*. [S.l.]: Springer Berlin Heidelberg, 2008. p. 1079–1099.

SANTOS, S. R. B. dos. *Planejamento e Controle para a Contrução Autônoma de Estruturas Tridimensionais utilizando Quadrirrotores*. Tese (Doutorado) — Instituto Tecnológico de Aeronáutica, 2014.

SANTOS, S. R. B. dos; GIVIGI, S. N.; NASCIMENTO, C. L. Autonomous construction of multiple structures using learning automata: Description and experimental validation. *IEEE Systems Journal*, Institute of Electrical and Electronics Engineers (IEEE), v. 9, n. 4, p. 1376–1387, dec 2015.

SENSE, I. R. *Intel Real Sense*. 2017. Disponível em: <https://www.intel.com/content/www/us/en/architecture-and-technology/realsense-overview.html>. Acesso em: 31 maio 2017.

SHARMA, S.; BISWAL, B. B.; DASH, P.; CHOUDHURY, B. B. Optimized robotic assembly sequence using ACO. In: *2009 Fifth International Conference on MEMS NANO, and Smart Systems*. [S.l.]: IEEE, 2009.

SINANOGLU, C.; RIZA, H. Assembly sequence planning using neural network approach. In: *Manufacturing the Future*. [S.l.]: Pro Literatur Verlag, Germany / ARS, Austria, 2006.

STEWART, R. L.; RUSSELL, R. A. A distributed feedback mechanism to regulate wall construction by a robotic swarm. *Adaptive Behavior*, v. 14, n. 1, p. 21–51, 2006. ISSN 10597123.

TANGO, G. *Google Tango*. 2017. Disponível em: <https://get.google.com/tango/>. Acesso em: 31 maio 2017.

THATHACHAR, M.; PHANSALKAR, V. Learning the global maximum with parameterized learning automata. *IEEE Transactions on Neural Networks*, Institute of Electrical and Electronics Engineers (IEEE), v. 6, n. 2, p. 398–406, mar 1995.

THATHACHAR, M. A. L.; SASTRY, P. S. *Networks of Learning Automata: Techniques for Online Stochastic Optimization*. [S.l.]: Springer US, 2004.

VRANCX, P.; VERBEECK, K.; NOWÉ, A. Networks of learning automata and limiting games. In: *Adaptive Agents and Multi-Agent Systems III. Adaptation and Multi-Agent Learning*. [S.l.]: Springer Berlin Heidelberg, 2008. p. 224–238.

WAWERLA, J.; SUKHATME, G. S.; MATARIC, M. J. Collective construction with multiple robots. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. [S.l.: s.n.], 2002. v. 3, p. 2696–2701 vol.3.

WEI, Y. Automatic generation of assembly sequence for the planning of outfitting processes in shipbuilding. *Journal of Ship Production and Design*, The Society of Naval Architects and Marine Engineers, v. 28, n. 2, 2012.

WERFEL, J.; PETERSEN, K.; NAGPAL, R. Designing collective behavior in a termite-inspired robot construction team. *Science*, American Association for the Advancement of Science (AAAS), v. 343, n. 6172, p. 754–758, feb 2014.

WILLMANN, J.; AUGUGLIARO, F.; CADALBERT, T.; DANDREA, R.; GRAMAZIO, F.; KOHLER, M. Aerial robotic construction towards a new field of architectural research. *International Journal of Architectural Computing*, SAGE Publications, v. 10, n. 3, p. 439–460, sep 2012.

WISMER, S.; HITZ, G.; BONANI, M.; GRIBOVSKIY, A.; MAGNENAT, S. Autonomous construction of a roofed structure: Synthesizing planning and stigmergy on a mobile robot. In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. [S.l.]: IEEE, 2012.

ZHANG, Z. Microsoft kinect sensor and its effect. *IEEE Multimedia*, Institute of Electrical and Electronics Engineers (IEEE), v. 19, n. 2, p. 4–10, feb 2012.

Apêndices

APÊNDICE A – ROS

Parâmetros do RTAB-Map

```
<?xml version="1.0" encoding="UTF-8"?>
<launch>
  <group ns="rtabmap">
    <node name="rtabmap" pkg="rtabmap_ros" type="rtabmap" output="screen" args="">
      <param name="frame_id" type="string" value="base_footprint" />
      <param name="subscribe_depth" type="bool" value="true" />
      <param name="odom_frame_id" type="string" value="/odom" />
      <remap from="rgb/image" to="/camera/data_throttled_image" />
      <remap from="depth/image" to="/camera/data_throttled_image_depth" />
      <remap from="rgb/camera_info" to="/camera/data_throttled_camera_info" />
      <param name="rgb/image_transport" type="string" value="compressed" />
      <param name="depth/image_transport" type="string" value="compressedDepth" />
      <param name="queue_size" type="int" value="100" />
      <!-- RTAB-Map's parameters -->
      <param name="cloud_decimation" type="int" value="5" />
      <param name="grid_eroded" type="bool" value="true" />
      <param name="grid_cell_size" type="double" value="0.01" />
      <param name="RGBD/NeighborLinkRefining" type="string" value="false" />
      <param name="RGBD/ProximityBySpace" type="string" value="true" />
      <param name="Reg/Strategy" type="string" value="0" />
      <param name="RGBD/AngularUpdate" type="string" value="0.1" />
      <param name="RGBD/LinearUpdate" type="string" value="0.1" />
      <param name="Mem/RehearsalSimilarity" type="string" value="0.45" />
      <param name="Mem/NotLinkedNodesKept" type="string" value="false" />
      <param name="Mem/ImageDecimation" type="string" value="1" />
      <param name="Rtabmap/StartNewMapOnLoopClosure" type="string" value="true" />
      <param name="Rtabmap/TimeThr" type="string" value="600" />
      <param name="Rtabmap/DetectionRate" type="string" value="0" />
      <param name="Mem/RawDescriptorsKept" type="string" value="true" />
      <param name="RGBD/LoopClosureReextractFeatures" type="string" value="false" />
      <param name="Mem/UseDepthAsMask" type="string" value="true" />
      <param name="Reg/Force3DoF" type="string" value="true" />
      <param name="Vis/EstimationType" type="string" value="1" />
      <param name="Bayes/PredictionLC" type="string" value="0.1 0.36 0.30 0.16 0.062 0.0151
      ↪  0.00255 0.00035" />
      <param name="Optimizer/Slam2D" type="string" value="true" />
      <param name="Optimizer/Iterations" type="string" value="1000" />
      <param name="RGBD/OptimizeFromGraphEnd" type="string" value="false" />
      <param name="Optimizer/Strategy" type="string" value="2" />
      <param name="Optimizer/Robust" type="string" value="false" />
      <param name="Optimizer/VarianceIgnored" type="string" value="true" />
      <param name="RGBD/PlanStuckIterations" type="string" value="10" />
      <param name="Kp/DetectorStrategy" type="string" value="0" />
      <param name="Kp/MaxFeatures" type="string" value="0" />
      <param name="Kp/MaxDepth" type="string" value="4.0" />
      <param name="Kp/MinDepth" type="string" value="0.65" />
      <param name="SURF/HessianThreshold" type="string" value="500" />
      <param name="Mem/BadSignaturesIgnored" type="string" value="true" />
      <param name="Vis/MinInliers" type="string" value="30" />
      <param name="Vis/InlierDistance" type="string" value="0.1" />
      <param name="Vis/MaxDepth" type="string" value="4.0" />
      <param name="Vis/MinDepth" type="string" value="0.65" />
      <param name="Vis/MaxFeatures" type="string" value="0" />
    </node>
  </group>
</launch>
```