PLANEJAMENTO DE MOVIMENTO DE MÚLTIPLOS ROBÔS COM RESTRIÇÃO DE COMUNICAÇÃO

RODRIGO WERBERICH DA SILVA MOREIRA DE OLIVEIRA

DISSERTAÇÃO DE MESTRADO EM ENGENHARIA DE SISTEMAS ELETRÔNICOS E AUTOMAÇÃO

DEPARTAMENTO DE ENGENHARIA ELÉTRICA

FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASÍLIA

UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

MULTIPLE ROBOT COMMUNICATION AWARE MOTION PLANNING
COMUNICAÇÃO

PLANEJAMENTO DE MOVIMENTO DE MÚLTIPLOS ROBÔS COM RESTRIÇÃO DE

RODRIGO WERBERICH DA SILVA MOREIRA DE OLIVEIRA

ORIENTADOR: PROF. DR. GEOVANY ARAÚJO BORGES
COORIENTADOR: PROF. DR. MARCELO MENEZES DE CARVALHO

DISSERTAÇÃO DE MESTRADO EM ENGENHARIA DE
SISTEMAS ELETRÔNICOS E AUTOMAÇÃO

PUBLICAÇÃO: PGEA.DM-740/20

BRASÍLIA/DF: JANEIRO - 2020

UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

PLANEJAMENTO DE MOVIMENTO DE MÚLTIPLOS ROBÔS COM RESTRIÇÃO DE COMUNICAÇÃO

RODRIGO WERBERICH DA SILVA MOREIRA DE OLIVEIRA

DISSERTAÇÃO DE MESTRADO SUBMETIDA AO DEPARTAMENTO DE ENGENHARIA ELÉTRICA DA FACULDADE DE TECNOLOGIA DA UNIVERSIDADE DE BRASÍLIA COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM ENGENHARIA ELÉTRICA.

APROVADA POR:

_____

Prof. Dr. Geovany Araújo Borges – ENE/Universidade de Brasília
(Orientador)

_____

Prof. Dr. João Yoshiyuki Ishihara – ENE/Universidade de Brasília
Membro Interno

_____

Prof. Dr. Mariana Costa Bernardes - FGA/Universidade de Brasília
Membro Externo

BRASÍLIA, 24 DE JANEIRO DE 2020.

FICHA CATALOGRÁFICA

DE OLIVEIRA, RODRIGO WERBERICH DA SILVA MOREIRA
Planejamento de movimento de múltiplos robôs com restrição de comunicação [Distrito Federal] 2020.
vii+95 p., 210 x 297 mm (ENE/FT/UnB, Mestre, Engenharia de Sistemas Eletrônicos e Automação, 2020).
DISSERTAÇÃO DE MESTRADO – Universidade de Brasília, Faculdade de Tecnologia.
Departamento de Engenharia Elétrica

1. Planejamento de movimento
2. Planejamento baseado em amostra
3. Restrição de comunicação
4. Robótica
I. ENE/FT/UnB
II. Título (série)

REFERÊNCIA BIBLIOGRÁFICA

DE OLIVEIRA, R. W. S. M. (2020). Planejamento de movimento de múltiplos robôs com restrição de comunicação, DISSERTAÇÃO DE MESTRADO, Publicação PGEA.DM-740/20, Departamento de Engenharia Elétrica, Universidade de Brasília, Brasília, DF, vii+95.

CESSÃO DE DIREITOS

AUTOR: Rodrigo Werberich da Silva Moreira de Oliveira
TÍTULO: Planejamento de movimento de múltiplos robôs com restrição de comunicação.
GRAU: Mestre ANO: 2020

Rodrigo Werberich da Silva Moreira de Oliveira

Departamento de Eng. Elétrica (ENE) - FT
Universidade de Brasília (UnB)
Campus Darcy Ribeiro
CEP 70919-970 - Brasília - DF - Brasil

Dedico esse trabalho a minha esposa Letícia
e aos meus pais Rui e Eliana.

AGRADECIMENTOS

Agradeço meus orientadores Professor Geovany e Professor Marcelo pela imensa paciência e atenção que eles tiveram comigo durante essa jornada. Agradeço aos meus pais Eliana e Rui por todo o esforço que eles tiveram em me criar e trazerem até onde estou. Agradeço a todos os meus irmãos Carol, Karen, Rhon e Carla por servirem de inspiração para mim. Agradeço a todos os meus amigos Camila, João Bosco, Alexandre, Artur, Thiago, Sarah, Ricardo, Rodrigo, Giordano, entre tantos outros, que me ajudaram a relaxar e descarregar as angústias quando necessário. E um grande obrigado a minha esposa Letícia, que não somente esteve comigo em todos os momentos de alegria e dificuldade deste trabalho, mas também me transformou no homem que eu sou hoje em dia, serviu de inspiração, me relaxou e consolou, entre tantas outras coisas. Amo muito todos vocês.

RESUMO

Título: Planejamento de movimento de múltiplos robôs com restrição de
Autor: Rodrigo Werberich da Silva Moreira de Oliveira
Orientador: Prof. Dr. Geovany Araújo Borges
Coorientador: Prof. Dr. Marcelo Menezes de Carvalho

Esse trabalho apresenta a proposta de uma arquitetura de planejamento de movimento com restrição de comunicação para o desenvolvimento de um sistema de assistentes laboratoriais para o LARA. A arquitetura proposta é distribuída entre um time de robôs e servidores na nuvem que lidam com cálculos mais computacionalmente pesados. Essa arquitetura é composta por três módulos principais: o planejamento de missão, a coordenação de missão e o executor de movimento. Os dois primeiros são módulos mais deliberativos e foram alocados para os servidores, enquanto que o executor é mais reativo e é alocado para cada um dos robôs. O foco deste trabalho foi o desenvolvimento de ferramentas que servissem como base para o planejamento e a coordenação de missão. O planejamento de missão é responsável por determinar os pontos chave que um robô deve passar para executar sua tarefa, e também definir qual tarefa deve ser executada em um dado momento. Para resolver a questão dos pontos chave, foi criado um algoritmo capaz de calcular tarefas-α. Esse termo foi criado neste trabalho e representa tarefas que visam auxiliar a execução de uma tarefa principal, fornecendo conexão para robôs que tenham que executar tarefas fora de uma região de conectividade. Elas permitem determinar um número mínimo de robôs que é necessário para realizar uma tarefa fora da zona de conectividade. Com o auxílio dessas tarefas-α foi proposta a criação de heurísticas que estimam o custo de execução de uma determinada tarefa sendo executada por múltiplos robôs em uma área que não possua conectividade. Utilizando essas heurísticas propôs-se um método para determinar qual tarefa escolher para executar. A coordenação de missão é realizada com o auxílio de algoritmos de busca em árvores aleatórias como o RRT e o RRT*, para garantir que o caminho executado seja realizável sem a perda de conexão adaptou-se o módulo de detecção de colisão para verificar situações onde haveria quebra de conexão. Os testes realizados apontaram algumas situações em que os métodos propostos podem apresentar problemas, mas foi possível constatar que essa arquitetura é uma arquitetura funcional e que esses métodos formam uma base sólida para implementação dessa arquitetura.

ABSTRACT

Title: *Multiple Robot Communication Aware Motion Planning* comunicação
Author: Rodrigo Werberich da Silva Moreira de Oliveira
Supervisor: Prof. Dr. Geovany Araújo Borges
Co-Supervisor: Prof. Dr. Marcelo Menezes de Carvalho

This works presents a Communication Aware Motion Planning architecture that aims to implement a laboratory assistant system for LARA. The proposed architecture is distributed between the team of robots and cloud server that must execute the most computationally heavy processes. This architecture is composed of three main modules: the mission planer, the mission coordinator and the movement executor. The first two modules are more deliberative and were allocated to the cloud servers, the executor module is more reactive and was allocated to each of the robots. The main focus of this work was the development of tools that would be useful to create a solid basis for the mission planning and coordination modules. The mission planning is responsible for determining the key points that a robot must go through in order to accomplish a given task. It is also responsible for defining which task must be executed given a set of tasks. To solve the key points issue, an algorithm capable of calculation α-task was created. This term was created during the development of this work an represent auxiliary tasks to a main task that aim to provide connection to robots that must execute tasks outside of the connectivity area. They allow you to determine the minimum amount of robots required to execute a task outside of the connectivity region. Using these α-tasks we created some heuristics to estimated the cost of executing a task outside of the connectivity area with multiple robots. Using these heuristics we proposed a method to choose which task must be executed. The mission coordination module is performed with the aid of random trees search algorithms, such as RRT and RRT*. To ensure that the path returned by these algorithms maintains connection, the obstacle detection module was adapted to also verify situations in which the robots would lose connection. The test performed showcased some situation in which the proposed methods may show some problems, but it was possible to state that this architecture is a valid one and that the proposed methods generated a solid ground for the implementations of this architecture.

1 INTRODUÇÃO 1
1.1 CONTEXTUALIZAÇÃO 1
1.2 DEFINIÇÃO DO PROBLEMA 2
1.3 OBJETIVO DO TRABALHO 4
1.4 RESULTADOS OBTIDOS 5
1.5 APRESENTAÇÃO DO MANUSCRITO 5

2 FUNDAMENTAÇÃO TEÓRICA 6
2.1 INTRODUÇÃO 6
2.2 REVISÃO BIBLIOGRÁFICA 6
2.2.1 ROBÔS COMO PONTO DE ACESSO 6
2.2.2 MANUTENÇÃO DE REDES DE CONEXÃO 7
2.2.3 PLANEJAMENTO DE MOVIMENTO COM RESTRIÇÃO DE COMUNICAÇÃO (CAMP) 7
2.3 *Communication-Aware Motion Planning* 9
2.4 ALGORITMOS DE PLANEJAMENTO 11
2.4.1 CONCEITOS BÁSICOS 11
2.4.2 PLANEJAMENTO DE MOVIMENTO 13
2.4.3 O ESPAÇO DE CONFIGURAÇÃO 15
2.4.4 PLANEJAMENTO BASEADO EM AMOSTRAS 17
2.4.5 BUSCAS EM ÁRVORES ALEATÓRIAS 20
2.4.6 ALGORITMO DIJKSTRA 27
2.4.7 TRIANGULAÇÃO DE DELAUNAY E FORMAS-α 28

3 PROPOSTA 31
3.1 INTRODUÇÃO 31
3.2 DETALHAMENTO E MODELAGEM DO SISTEMA 31
3.2.1 LINGUAGEM DESCRITIVA DE PROBLEMAS DE CAMP 36
3.3 CALCULANDO AS TAREFAS AUXILIARES: TAREFAS-α 43
3.4 DECIDINDO QUAL TAREFA EXECUTAR 49
3.5 CRIANDO UM PLANO DE EXECUÇÃO 53

4 RESULTADOS 62
4.1 INTRODUÇÃO 62
4.2 TAREFAS-α 62
4.3 COMPARANDO HEURÍSTICAS 65
4.4 SIMULANDO O SG11 69

5 CONCLUSÃO 77
5.1 CONCLUSÃO 77
5.2 TRABALHO FUTUROS 78

A EXEMPLO DE UMA MODELAGEM CAMP 80

BIBLIOGRAFIA 91

Lista de Figuras

1.1 As camadas da arquitetura IoRT 2
1.2 Os robôs Pioneer do Laboratório de Automação e Robótica (LARA). 3

2.1 Modelos de comunicação 10
2.2 Exemplo de problema de planejamento 13
2.3 Espaço de estados 17
2.4 A filosofia por trás do planejamento baseado em amostras. Imagem traduzida de [43]. 17
2.5 Exemplo de um algoritmo de planejamento baseado em amostras com busca e amostragem incrementais 19
2.6 Exemplo de um algoritmo de planejamento baseado em amostras com *roadmaps*. 20
2.7 RRT após 1000 iterações 21
2.8 "Armadilha de inseto". Imagem traduzida de [43]. 22
2.9 Triangulação de Delaunay (em azul), e seu Diagrama de Voronoi (em preto) correspondente, de um conjunto de pontos (em laranja). Adaptado de [55]. 29
2.10 Formas-α de um conjunto de pontos. Fonte [58]. 30

3.1 Diagrama de execução das diferentes etapas do planejamento. Usuários podem criar novas missões. Um servidor na nuvem prepara e coordena a missão, que é executada pelo robôs. 34
3.2 Mapa construído a partir de modelo probabilístico do canal, baseado em Campos Gaussianos aleatórios. As cores de fundo representam a intensidade do sinal em uma determinada região, de acordo com a legenda na parte superior direita da imagem. O trajeto pontilhado representa o trajeto que o robô fez durante o mapeamento, e o ponto branco representa a posição real do roteador. Fonte [41]. 35
3.3 Estrutura de um JSON que representa um problema de CAMP. 37
3.4 JSON dos tipos de robôs. 37
3.5 JSON dos tipos de tarefas. 38
3.6 JSON dos tipos de tarefas. 38
3.7 JSON da função de recompensa. 38
3.8 JSON de um robô. 39
3.9 JSON de um tarefa. 40
3.10 JSON de um missão. 40
3.11 JSON de uma área de busca. 41
3.12 JSON da área obstruída. 42
3.13 Um cenário onde uma tarefa se encontra dentro, enquanto a outra se encontra fora da zona de conexão. Como determinar se é possível realizar a segunda tarefa? 43
3.14 Arranjos de conexão para realização de uma tarefa. 47
3.15 Simplificação do conjunto conjunto base $\mathbf{T}'_\alpha$ para o conjunto final $\mathbf{T}_\alpha$ 49
3.16 Todas as tarefas que precisam ser realizadas para que a missão m esteja completa. 50
3.17 Visualizando arestas em $\mathcal{C}$. 56
3.18 Um exemplo de marcação de conectividade para um conjunto $V_{\mathfrak{C}}$. As amostras em verde estão dentro da zona de conectividade. 57

3.19 Exemplo de uma situação que representa o problema da Figura 3.18. O retângulo preto é um obstáculo. 58

4.1 Os primeiros dois cenários de teste para as tarefas-α. Os pontos verdes representam os pontos de amostragem utilizados para construir o espaço de busca. Os segmentos de reta amarelos são a triangulação de Delaunay que conecta esses pontos. O retângulo vermelho representa um obstáculo. O círculo verde é a zona de conectividade real, enquanto que os pontos laranjas representam a estimação dessa zona de conectividade. As cruzes pretas são as tarefas ou tarefas-α que foram calculadas. As linhas azuis conectam as tarefas-α em sequência até a tarefa principal, mostrando a visibilidade entre as tarefas. 63
4.2 Cenário c: Uma passagem muita estreita entre a região onde se encontra a tarefa e a região de conectividade. 63
4.3 Os últimos dois cenários de teste para as tarefas-α 64
4.4 Um cenário com três missões (tarefas em vinho, verde e azul) e três robôs. 66
4.5 Um modelo de parte do primeiro andar do SG-11. A parte inferior da figura é o laboratório LARA. 69
4.6 Posicionamento das tarefas e dos robôs. Tarefas que pertencem a mesma missão possuem a mesma cor. 71
4.7 Tarefas-α para as missões atuais do SG-11 72
4.8 Plano gerado pelo coordenador de missão e uma visualização da execução dos três primeiros passos. 73
4.9 Execução dos últimos quatro passos do planejamento feito pelo coordenador de missão. 74
4.10 Plano de execução para a tarefa t_4 75

A.1 Representação gráfica do arquivo descritor em JSON. A descrição retrata uma parte do primeiro andar do SG 11, em que a parte inferior serio o LARA. 80

Lista de Tabelas

3.1 Representação em forma de tabela das aresta que possuem ou não visibilidade para a Figura 3.19. 60

4.1 Número de robôs necessários para realizar as tarefas dos cenários apresentados nas Figuras 4.1 a 4.3. 65
4.2 Valores dos custos totais, prazos e folgas para cada uma das missões utilizando as diferentes heurísticas. 67
4.3 Tempos médios, em μs, de execução de 100 iterações, para o cálculo dos custos de cada uma das missões. 68
4.4 Auxílio para compreender o plano da Figura 4.8a. Cada linha representa uma mudança de estado, e o número indica em qual posição da lista de posições daquele robô ele deve estar naquele estado. 74
4.5 Auxílio para compreender o plano da Figura 4.10. Cada linha representa uma mudança de estado, e o número indica em qual posição da lista de posições daquele robô ele deve estar naquele estado. 76

Símbolos Latinos

$\mathfrak{P}$	Plano ou caminho
X	Espaço de estados
$\mathcal{C}$	Espaço de configuração
$\mathcal{W}$	Ambiente ou "mundo"
$\mathcal{W}_O$	Área obstruída do ambiente
$\mathcal{A}$	Corpo rígido de um robô
q	Uma configuração qualquer
$\mathcal{C}_{obs}$	Espaço de configuração obstruído
$\mathcal{C}_{free}$	Espaço de configuração livre
q_i	Configuração inicial
q_f	Configuração final
$\mathcal{G}$	Grafo
V	Vértices ou nós de um grafo
E	Arestas de um grafo
M_C	Mapa de custo
$\mathtt{T}$	Árvore de busca
$\mathcal{D}$	Mapa de custo de Dijkstra
$\mathbf{R}$	Um conjunto de robôs
r_i	i-ésimo robô
r_τ^i	Tipo do i-ésimo robô
r_p^i	Posição do i-ésimo robô
$\mathbf{M}$	Um conjunto de missões
m_j	j-ésima missão
$m_{\mathbf{T}}^j$	Conjunto de tarefas que fazem parte da j-ésima missão
m_π^j	Prioridade da j-ésima missão
$m_\mathfrak{d}^\mathfrak{j}$	Prazo da j-ésima missão
Π	Níveis de prioridade existentes
$\mathbf{T}$	Um conjunto de tarefas
t_k	k-ésima tarefa
$t_{\mathbf{T}}^k$	Conjunto de tarefas pré requisitos da k-ésima tarefa
t_p^k	Posição da k-ésima tarefa
t_τ^k	Tipo da k-ésima tarefa
$\mathbf{e}$	Função de esforço
$\mathbf{r}$	Função de recompensa
$\mathcal{W}_\mathcal{B}$	Área de busca
$\mathfrak{C}$	Função de conectividade
$\mathcal{M}$	Mapa de busca
$\mathcal{P}_{con}$	Pontos da borda da zona de conectividade
r_{dm}	Raio de distância mínima entre amostras
$\mathcal{P}$	Pontos de uma amostragem

$\mathcal{O}$	Função de obstrução
$\mathcal{P}_{free}$	Pontos da amostragem que pertencem a $\mathcal{C}_{free}$
$h_{\mathfrak{C}}$	Custo entre duas configurações
$\mathcal{N}_{\alpha}(t)$	Número de tarefas-α associados a uma tarefa t
$h_1, h_2, \ldots$	Heurísticas de custo
$\alpha_i t$	i-ésima tarefa-α de t
$\mathfrak{f}$	Folga de uma missão
$\mathbf{r}_m$	Recompensa de uma missão
Q	Fila de processamento

Símbolos gregos

ρ	Função de distância
α	Função de amostragem
λ	Um limiar
δ	Um pequeno intervalo
$\mathcal{T}_R$	Conjunto de tipos de robôs
$\mathcal{T}_{\mathbf{T}}$	Conjunto de tipos de tarefas
$\gamma_{\mathfrak{C}}$	Limiar mínimo de conexão
$\rho_{\pi k}$	Função de distância com peso para zona de conectividade
$\Phi_{\pi k}$	Função de distância com peso para zona de conectividade considerando obstáculos
Ψ	Porcentagem do caminho dentro da zona de conectividade

Siglas

IoT	*Internet of Things* (Internet das coisas)
IoRT	*Internet of Robotic Things* (Internet das coisas robóticas)
CAMP	*Communication Aware Motion Planning*
LARA	Laboratório de Automação e Robótica
RSSI	*Received signal strength indication* (Indicador de intensidade do sinal recebido)
LTL	Lógica Temporal Linear
RCAMP	*Resilient Communication Aware Motion Planning*
VSM	*Vertex Selection Method*
LPM	*Local Planning* Method
RDT	*Rapidly exploring Dense Tree* (Árvore densa de exploração rápida)
RRT	*Rapidly exploring random trees* (Árvore aleatória de exploração rápida)
RRT*	Uma versão assintoticamente ótima do RRT
UnB	Universidade de Brasília
SLAM	*Simultaneous Location and Mapping*

1 INTRODUÇÃO

1.1 CONTEXTUALIZAÇÃO

Desde a década de 1990 as áreas de robótica e de engenharia de redes já apresentavam muitas pesquisas em comum, com um certo interesse de alguns estudiosos dessa época, e do início dos anos 2000, em estudar maneiras de controlar robôs por meio de telemetria através da internet [1, 2, 3]. Mais recentemente, os avanços das redes 5G de telefones, de carros autônomos e a nova onda de entusiasmo relacionado a Internet das coisas (IoT), têm levantado muitas discussões em torno da interseção dessas áreas.

Os trabalhos em [4] e [5] propõem arquiteturas teóricas para sistemas de redes 5G de carros autônomos. O maior intuito em criar redes veiculares é promover um aumento na segurança, aliado com muitas outras aplicações que podem aproveitar desse ecossistema para melhorar a eficiência do tráfego veicular e prover diversos serviços de informação e entretenimento [4]. Uma demonstração prática dos benefícios de redes de carros conectados pode ser encontrada em [6].

O conceito de redes colaborativas de veículos e objetos em geral, instigado pela onde de IoT alcançou a robótica criando o termo Internet das coisas robóticas (IoRT) [7] . Esse conceito foi cunhado por P.P. Ray e une a robótica em nuvem com a Internet das coisas (IoT). A ideia básica do conceito de IoT é que a presença constante de uma variedade de coisas ou objetos no nosso meio ambiente, que por meio de um esquema de endereçamento único são capazes de interagir e cooperar entre si, gere uma sinergia e faça com que eles consigam trabalhar em conjunto para alcançar um objetivo em comum.

Incorporar o conceito de IoT à robótica permite que esses ambientes extremamente controlados no qual os robôs devem trabalhar possam ser expandidos por meio da incorporação dos mesmos como uma das “coisas” do sistema. Com o robô conectado nessas redes, ele pode obter informações do ambiente no qual ele se encontra de maneira fácil e intuitiva.

Dessa forma, IoRT é definido como uma infraestrutura que disponibiliza serviços de robótica avançados por meio da interconexão das coisas robóticas. Sendo baseado em tecnologias de comunicação e troca de informação já existentes, nas quais computação e armazenagem em nuvem e outras tecnologias disponíveis na internet estão centradas em torno dos benefícios da infraestrutura convergida da nuvem e dos serviços compartilhados. Estes permitem aos robôs aproveitarem os benefícios de recursos poderosos de processamento, armazenagem e comunicação de dados modernos associados à nuvem.

Um exemplo de de arquitetura de IoRT em [7] propõem a existência de cinco camadas (Figura 1.1): camada de hardware, na qual ficam as implementações físicas dos robôs e das “coisas”; a camada de rede, na qual se encontram as tecnologias de comunicação; a camada de Internet, que é a camada principal, na qual se encontram os protocolos de comunicação; a camada de infraestrutura, que é uma camada na qual ocorre um conglomerado de plataformas e serviços, como por exemplo, serviço de nuvem e plataformas de suporte a robótica como o ROS; e a camada de aplicação, que disponibiliza ao usuário os diferentes serviços que plataformas robóticas podem oferecer.

Todos esses benefícios vêm com um custo adicional de infraestrutura de conectividade, não somente entre os robôs, mas também entre robôs e centros de computação em nuvem. Porém, acredita-se que esses custos são compensados pelos benefícios. Um sistema em rede facilita muito a colaboração entre as partes. Ele

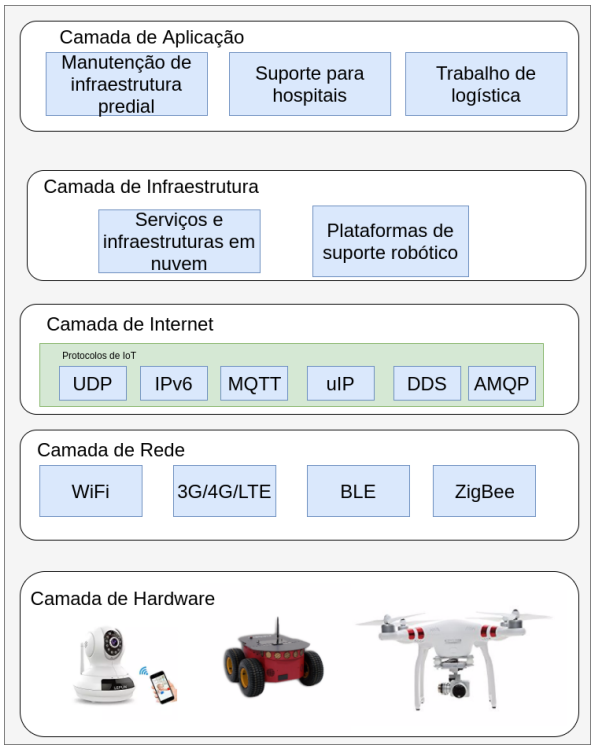


Figura 1.1: As camadas da arquitetura IoRT

aumenta também a sua robustez, pois caso um falhe, outro pode assumir. Outra coisa que é facilitada é o reaproveitamento de partes. Com um sistema em nuvem, é possível que robôs fiquem mais leves e baratos, podendo dedicar seus processamento mais imediato para lidar com situações emergenciais e focar na coleta de dados, repassando processamentos mais pesados para computadores com um maior poder de processamento e de armazenagem.

A manutenção desse tipo de infraestrutura de conexão, enquanto os robôs executam suas tarefas, tem sido também um grande interesse de pesquisa, como será abordado posteriormente na pesquisa bibliográfica. Algoritmos que abordam estratégias relacionadas a isso receberam a alcunha de planejamento de movimento com restrições de comunicação, conhecido também como *Communication Aware Motion Planning* (CAMP). Acredita-se que uma das primeiras menções a esse termo foi feita em [8]. Essa área será o foco do desenvolvimento deste trabalho.

Esse tipo de infraestrutura também pode auxiliar times de robôs em situações de busca e resgate, permitindo, por exemplo, que redes de controle por telemetria, como a do trabalho [9], alcancem regiões mais distantes e de difícil acesso. Ou então, para o estabelecimento de redes de comunicação em zonas atingidas por desastre por meio do uso de Drones [10]. Entre outras aplicações, como as apresentadas em [11].

1.2 DEFINIÇÃO DO PROBLEMA

A comunicação é cada vez mais imprescindível para o controle, monitoramento e colaboração de robôs, não apenas entre eles, mas também com os seres humanos que desejam que esses sistemas robóticos executem

algo para eles.

Com o intuito de criar um ambiente de desenvolvimento de tecnologias ligadas a robótica e ao campo de planejamento de movimento com restrições de comunicação, analisou-se o trabalho em [12], que foi posteriormente compilado com outros projetos complementares para a escrita do artigo [13] que explica uma arquitetura autônoma para robôs móveis voltada para o cenário de competições de robótica. Dele saiu a ideia de criar um arquitetura robusta de CAMP.

Esse arcabouço deve fornecer uma metodologia para manter uma conexão estável de um grupo de robôs heterogêneos com uma ou mais bases fixas. Para demonstrar a capacidade do arcabouço e incentivar outras pesquisas, decidiu-se criar um sistema de assistentes laboratoriais com os robôs Pioneers, Figura 1.2, disponíveis no Laboratório de Automação e Robótica (LARA). Esses robôs devem receber de um operador humano, ou de outro sistema, um conjuntos de tarefas que devem ser executadas em um determinado tempo.

Os robôs terão capacidades diferentes, um deles terá a capacidade de levar café para as pessoas do laboratório, o outro buscará impressões da impressora e levará para as pessoas e o último é responsável por pegar materiais do almoxarifado e trazê-los de volta para uma área de depósito. Algumas dessas capacidades podem ser manifestadas em mais de um robô, por exemplo, pode ser que dois deles sejam capazes de pegar objetos no depósito.

Todas essas requisições seriam feitas pelos alunos, professores e pesquisadores do laboratório em um site, que estaria hospedado em um computador no laboratório que estaria conectado em um roteador que é considerado a base dos robôs. Os robôs recebem as suas tarefas por meio da rede desse roteador e têm que gerenciar tanto a execução de suas tarefas, quanto garantir que sempre exista uma maneira de mensagens destinadas a um dos robôs específicos alcançá-los.

Os robôs devem, ao mesmo tempo, decidir a melhor maneira de executar as tarefas recebidas ou determinar que uma tarefa específica não pode ser realizada. Para isso, o sistema deve levar em conta a conectividade do sistema, isso significa que, a todo momento deve existir algum caminho de conexão entre qualquer um dos robôs e uma das bases. Garantindo assim o monitoramento e controle do sistema o tempo todo. O sistema

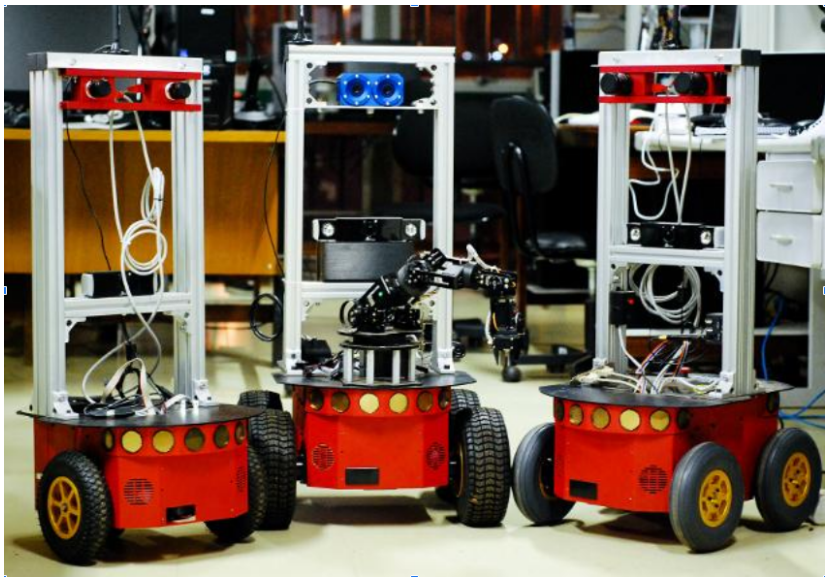


Figura 1.2: Os robôs Pioneer do Laboratório de Automação e Robótica (LARA).

também deve garantir o tempo de execução de cada tarefa e qual tipo de tarefa deve ser executada.

O ambiente no qual essas tarefas se encontra é parcialmente conhecido geograficamente mas é desconhecido em relação aos canais de comunicação. Os robôs devem ser capazes de gerar planos que permitam a realização dessas tarefas nesse ambiente da maneira mais rápida possível e apontar tarefas que não são possíveis de realizar.

Os problemas de navegação e localização serão considerados como resolvidos e estão fora do escopo deste trabalho, abrindo espaço para integração com outros trabalhos e pesquisas do LARA.

1.3 OBJETIVO DO TRABALHO

O objetivo deste trabalho é propor uma arquitetura de planejamento de movimento com restrição de movimento capaz de implementar um sistema cooperativo de assistentes laboratoriais no LARA que tenha suporte para assistência, monitoramento, telemetria e integração com outros sistemas de IoT e de computação em nuvem. E além disso, fornecer um conjunto de ferramentas bases para implementação desse sistema. Em vista disso, propôs-se um arquitetura composta por três módulos: o planejamento de missão, a coordenação de missão e a execução do movimento. O planejamento de missão é responsável por guardar uma fila das requisições que são recebidas pelo sistema, analisá-las e decidir qual missão o conjunto de robôs deve executar agora. Uma missão é um conjunto de tarefas que os robôs devem executar para cumprir um requisição. Ademais, o planejamento deve fornecer um conjunto de pontos chave pelo qual os robôs devem passar para executar cada uma das tarefas da missão. A coordenação recebe a missão escolhida pelo planejador, juntamente com os seus pontos chave, e gera uma rota que os robôs devem executar para cumprir a missão sem colidir com obstáculos ou perder a conexão. A execução do movimento é responsável por colher informações do ambiente, reagir a mudanças no ambiente e criar planos locais que permitam a execução do plano gerado pelos módulos anteriores em um ambiente real e dinâmico.

Decidiu-se implementar essa arquitetura de maneira distribuída, com os módulos de planejamento e coordenação de missão sendo alocados para servidores de computação na nuvem, enquanto que a execução é responsabilidade de cada um dos robôs do sistema. Essa escolha foi feita pois o planejamento e coordenação são processos pesados que poderiam interferir na capacidade de processamento dos robôs individuais. Esses processos podem ser paralelizados a execução do movimento, gerando novos planos enquanto os robôs executam os anteriores. Além disso, máquinas mais potentes podem conseguir refinar melhor os planejamentos. Outro benefício dessa separação é que os robôs podem ter acessos a outras ferramentas e serviços que podem estar disponíveis na nuvem, como, por exemplo, um serviço de mapeamento coletivo entre os robôs, aderindo aos princípios de IoRT e robótica em nuvem.

Por questões de tempo limitante, decidiu-se por focar na parte da arquitetura que é realizada por meio de computação em nuvem. Os robôs Pioneers do laboratório não estavam completamente operantes, e apesar de que vários reparos tenham sido realizados nos robôs para permitir a implementação do sistema neles, optou-se por descontinuar essa tentativa, dado que o tempo extra que teria que ser gasto para terminar os reparos e implementar um sistema de navegação minimamente funcional para os objetivos deste trabalho, inviabilizariam o mesmo.

1.4 RESULTADOS OBTIDOS

Os principais resultados deste trabalho foram a publicação do artigo *A robot architecture for outdoor competitions* na revista *Journal of Intelligent & Robotic Systems* e o desenvolvimento de três ferramentas para a implementação dos módulos de planejamento de missão e de coordenação de missão. A primeira é o algoritmo de cálculo de tarefas-α, tarefas auxiliares que servem tanto como apoio para próximas etapas do planejamento, dizendo posicionamentos que os robôs devem manter para realizar tarefas sem perda de conexão, como uma ferramenta capaz de dizer se uma determinada tarefa é realizável no ambiente atual com o número de robôs disponível respeitando as restrições de comunicação. A segunda é uma ferramenta que utiliza heurísticas para decidir qual dessas tarefas é mais custosa e determinar qual delas deve ser executada em cada momento. A última é uma ferramenta capaz de utilizar as tarefas-α de uma determinada tarefa e transformá-las em um plano executável.

Os testes executados permitiram a constatação de que essas ferramentas, de fato, fornecem uma boa base de desenvolvimento para a implementação do sistema proposto. Entretanto, eles também indicaram algumas situações em que as técnicas desenvolvidas podem falhar, gerando o ferramental necessário para melhorá-las e expandir o sistema.

1.5 APRESENTAÇÃO DO MANUSCRITO

Este trabalho está organizado em cinco capítulos e um apêndice. O Capítulo 1 é responsável por apresentar e contextualizar o trabalho desenvolvido, definir o problema que se deseja atacar e os objetivos e apresentar um resumo dos resultados obtidos.

O Capítulo 2 contém a fundamentação teórica para a compreensão dos conceitos básicos necessários para este trabalho. Ele é composto por quatro seções. Ele inicia com uma breve introdução ao capítulo, seguido de uma revisão bibliográfica, apresentando os principais trabalhos da área. Depois ele familiariza o leitor com o CAMP. Por último, ele apresenta em detalhes os conceitos de algoritmos de planejamento que serão essenciais para a compreensão do restante deste trabalho, como espaços de configuração, planejamento baseado em amostras, algoritmo de Dijkstra, triangulação de Delaunay e formas-α.

O Capítulo 3 é dividido em cinco seções. Novamente uma pequena introdução, seguida de uma maior detalhamento e modelagem do sistema. Em seguida é apresentado o algoritmo de cálculo de tarefas-α. A próxima sessão explica o algoritmo de decisão de tarefas e apresenta as heurísticas utilizadas para isso. E por fim, é explicado a modelagem feita do problema para que ele possa ser planejado utilizando árvores de busca aleatórias.

O Capítulo 4 apresenta quatro seções. Uma introdução seguida dos quatro tipos de testes realizados. Os primeiros testes focaram em analisar o algoritmo de tarefas-α. O segundo faz uma comparação entre as heurísticas propostas. E o último foca na análise do planejamento por meio árvores aleatórias proposto.

O Capítulo 5 é o último, e apresenta as conclusões e propostas de trabalhos futuros. Ele é seguido do Apêndice A que mostra um exemplo de modelagem CAMP utilizando a linguagem descritiva proposta neste trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 INTRODUÇÃO

Neste capítulo serão introduzidos os conceitos básicos necessários para um melhor entendimento do trabalho aqui desenvolvido. A seção 2.2 vai expor uma breve revisão bibliográfica da intersecção dos campos da robótica e redes de comunicação, focando em alguns dos casos mais comuns dessa intersecção: robôs como ponto de acesso, manutenção de redes de conexão e planejamento de movimento com restrição de comunicação, do inglês *Communication-Aware Motion Planning* (CAMP). Posteriormente, a seção 2.3 explicará mais esse conceito de CAMP, comentando principalmente as maneira de categorizar esse campo. A seção 2.4 apresentará os conceitos básicos e o que é o planejamento de movimento, além de apresentar o espaço de configuração, como funciona o planejamento baseado em amostras e as principais técnicas necessárias para entender o planejamento de movimento utilizado neste trabalho. Além disso, são apresentados o algoritmo de Dijkstra e é feita uma breve introdução dos conceitos de triangulação de Delaunay e de formas-α. Acredita-se que feita a leitura desse capítulo e de suas referências, o leitor estará apto a entender o trabalho desenvolvido nos capítulos posteriores.

2.2 REVISÃO BIBLIOGRÁFICA

A intersecção entre robótica e redes sem fios forma uma área do conhecimento muito abrangente composta por diversas subáreas como: redes de comunicação com auxílio de robôs; gerenciamento e manutenção de conectividade e performance para times de robôs; equipes de resgate robóticas; *CAMP*; localização em ambientes fechados; casas inteligentes e robôs; entre muitas outras. O que se segue é uma breve revisão bibliográfica com uma parte da literatura das subáreas mais relevantes para esse trabalho.

2.2.1 Robôs como ponto de acesso

A ideia de utilizar robôs como pontos de acesso em sistemas de comunicação não é nada recente, com artigos falando sobre esse assunto desde antes de 2005, como por exemplo o trabalho em [14], que apresenta um algoritmo que lida com o problema de cobrir uma área (sem mapa ou sistema de localização externo) e realizar a distribuição de aparelhos capazes de montar uma rede que auxilia a tarefa sendo executada. O robô posteriormente também utiliza essa rede para auxiliar a sua navegação.

Mesmo não sendo novidade, esse tema ainda é discutido em publicações mais recentes, como, por exemplo, em [15, 16] é estudado o problema de criar e manter uma rede de comunicação com robôs considerando os clientes que utilizam essa rede e os obstáculos presentes no ambiente. É proposto um algoritmo de campo potencial para solucionar o problema. Esse campo é calculado a partir de um mapa, as posições dos clientes nesse mapa e uma qualidade mínima de taxa de comunicação para calcular os campos potenciais. Esse trabalho posteriormente foi expandido em [17].

Em um trabalho de 2017 [18] são propostos dois algoritmos: um para o posicionamento de aparelhos de comunicação e outro para estimar a localização dos mesmos por meio do indicador de intensidade do sinal recebido - RSSI.

2.2.2 Manutenção de redes de conexão

Outro aspecto muito importante é a gestão, manutenção e expansão dessas redes que não precisam estar limitadas a servir como infraestrutura de pontos de acesso, mas também para permitir a comunicação entre os robôs de um time que trabalham juntos para realizar uma tarefa. Existem estudos desde 2006 como [19], no qual o autor faz um estudo experimental de técnicas que buscam manter comunicação ponto a ponto em uma rede de robôs. Os testes foram feitos utilizando quatro robôs móveis e uma estação base em ambientes externos. Os testes avaliam e relacionam a relação sinal ruído, a distância, as velocidades reais e taxas de transmissão entre os pontos. A estrutura de comunicação é constante durante os testes, não ocorrendo roteamento das informações.

Os trabalhos em [20, 21] comparam quatro algoritmos de roteamento para redes Ad-Hoc por meio de testes em robôs reais em ambientes externos. São testados dois protocolos reativos (AODV e DSR) e dois protocolos proativos (OLSR e B.A.T.M.A.N). Os autores analisam principalmente os quesitos de perda de pacotes e de tempo para reestabelecer uma rota perdida. Os testes são realizados com até seis nós, sendo um o operador dos robôs e quatro robôs móveis, o autor não deixa claro qual seria o sexto nó.

Um algoritmo de campo potencial é utilizado em [22] para manter a conexão entre um robô, que tem uma tarefa para realizar, e sua base. Para isso os robôs são divididos em três categorias. Os robôs de tarefa, que tentam alcançar um alvo, os robôs seguidores, que seguem os robôs de tarefa até ser necessário eles se tornarem o último tipo robô, os robôs de roteamento, que ficam se posicionando de maneira a proporcionar a comunicação.

Algoritmos de expansão de redes, manutenção de formação e roteamento são propostos em [23]. Redes neurais são usadas em [24, 25] para manter uma cobertura de comunicação, é proposta uma rede neural que calcula uma formação para os robôs, a partir dos dados de RSSI dos robôs adjacentes, de forma a manter o RSSI sempre acima de um limiar especificado. Essa rede é previamente treinada e depois implementada nos robôs.

Em [26] é analisado um algoritmo que lida com o problema de planejar o movimento de um conjunto de robôs que servem como pontos de acesso para maximizar a comunicação entre um grupo estático de geradores de dados.

Uma análise sobre a importância de roteamento realizado por equipes de robôs, assim como uma análise sobre seus desafios e problemas, pode ser encontrada em [27].

2.2.3 Planejamento de movimento com restrição de comunicação (CAMP)

Uma das primeiras menções ao termo CAMP foi feita em [8] por Ghaffarkhah, A. e Mostofi, Y. Nesse trabalho é proposto um *framework* de planejamento de movimento que leva em consideração a comunicação para aumentar a probabilidade que o robô mantenha sua conexão com uma estação fixa, enquanto realiza uma tarefa de sensoriamento, em um ambiente realista de comunicação. Os autores expandiram seu trabalho em [28] no qual se deseja resolver o problema de um robô cujo objetivo é identificar a posição de certos alvos e

comunicar essa informação a uma base fixa remota. Dois cenários são analisados, um no qual a comunicação deve ser sempre mantida, informando a posição do robô a todo momento com a base fixa, e um outro cenário, no qual o robô pode se desconectar, encontrar o alvo, voltar a uma local que possuía comunicação e reportar a posição. Para resolver o primeiro cenário, eles utilizam as técnicas propostas em [8]. Os testes de ambos foram realizados apenas em simulação.

Em [29] é considerado um cenário no qual se deseja otimizar os custos de locomoção e transmissão para um robô que deve enviar uma quantia fixa de bits para uma estação, enquanto se desloca em uma trajetória pré definida. É elaborada uma arquitetura que permite o robô planejar sua velocidade, taxa de transmissão de dados e suas paradas baseado na predição probabilística da qualidade do sinal na trajetória.

Os trabalhos em [30, 31, 32] propõem um controlador que leva em consideração aspectos chamados por eles de cibernéticos. Esses aspectos consideram variáveis de roteamento de forma a maximizar a probabilidade de se ter uma rede conectada em determinadas posições. Eles também consideram aspectos físicos para determinar possíveis trajetórias que mantenham essas taxas acima de um limiar mínimo. Além disso, são empregados planejadores locais e globais para calcular o percurso dos robôs. Eles realizaram testes com cinco robôs analisando a taxa de comunicação entre eles.

Uma evolução de [29] foi realizada em [33], nessa abordagem, a trajetória não está mais pré definida e deseja-se, dado um conjunto de pontos de interesse, calcular a trajetória ótima (em que ordem visitar esses pontos), em relação a economia de movimento e de energia de comunicação. Já em [34], outra evolução de [29], na qual o problema é tratado com uma abordagem de tempo contínuo. A aceleração é considerada como a entrada, ao invés da velocidade, e também como uma medida do consumo de energia de locomoção. É usada então uma estrutura de otimização para minimizar os custos.

O CAMP também pode ser utilizado no contexto de cobrir da melhor maneira possível uma região para detectar eventos que possam estar acontecendo nessa área utilizando um time de robôs [35]. As preocupações são a qualidade de comunicação, o roteamento da informação e a detecção de eventos, de forma a que ela possa ser retransmitida para um par de estações de acesso fixo. O problema de mobilidade é resolvido por um programa de concavidades sequenciais distribuída.

Mais um evolução de [29] é apresentada em [36], porém dessa vez o robô tem um destino e deve escolher o caminho a percorrer, sua aceleração e sua taxa de transmissão durante o percurso, de forma a minimizar o consumo de energia respeitando uma série de restrições de energia e comunicação. Duas soluções são apresentadas: uma offline que considera que todas as medidas do canal são tomadas antes da execução do caminho; e outra online, na qual o robô toma as medidas enquanto anda.

Outra alternativa para realizar o CAMP é com uma perspectiva de teoria dos jogos [37]. Um problema de uma rede de sensores formada por robôs, cuja tarefa principal é coletar informações, pode ter seus planejamento de caminhos modelado como um jogo não cooperativo, no qual funções de utilidade são derivadas para cada robô e otimizadas localmente.

Um outro método para fazer o planejamento de movimento com restrição de comunicação pode ser visto em [38]. Ele é dividido em três partes: um planejador que considera as trajetórias que mantém a comunicação; um planejador que reutiliza robôs que já cumpriram sua missão para manter a conexão e um alocador de tarefas para garantir que os robôs farão em conjunto a melhor tarefa. Para considerar a conexão é levado em conta o RSSI.

A existência de uma rede de robôs que precisa realizar uma série de tarefas em um ambiente complexo, enquanto garante uma conexão ponta a ponta é uma aplicação muito comum em CAMP. Como em [39] que

faz isso com uma infraestrutura fixa de pontos de acesso. Tarefas são associadas a locais, são enunciadas sequencialmente e não são informadas a priori para os robôs. A informação obtida na tarefa é enviada de volta ao ponto de acesso por meio de uma rede *multi hop*. É proposto um esquema de controle híbrido e distribuído que monta uma rede em árvore que cresce dinamicamente. Para isso os robôs trocam constantemente de papel em relação a sua função na rede para otimizar as variáveis de comunicação e o planejamento de movimento em ambientes complexos. É considerado que existe um número infinito de robôs.

O tema também pode ser abordado de maneira intermitente, e não contínua, como em alguns dos trabalhos já citados. Isso quer dizer que não é necessário conexão à todo tempo. [40] lida justamente com isso, no qual um grupo de robôs ficam se deslocando na beira de uma grafo de comunicação e devem se encontrar de tempo em tempo para se comunicar. É apresentado um algoritmo que faz o planejamento de movimento de forma a garantir esses encontros usando LTL (Lógica Temporal Linear). Além disso, é apresentado um resolvedor para os conflitos que podem ser gerados por essa lógica.

[41] introduz o RCAMP - *Resilient Communication-Aware Motion Planner*, que foca no reestabelecimento autônomo de conexão de robôs que se perderam. Isso é feito por meio de um método de mapeamento *online* de sinais de rádio utilizando campos gaussianos aleatórios e com uma estratégia de reparo que leva em consideração tanto a conectividade quanto o objetivo quando se deslocando em direção a um local de conectividade, após ter ocorrido a perda da mesma.

A literatura da área é bem vasta, porém os trabalhos aqui apresentados devem ser capazes de dar um bom panorama do tema ao leitor, que, caso deseje, pode posteriormente se utilizar destes como ponto inicial para aprofundar sua pesquisa.

2.3 *COMMUNICATION-AWARE MOTION PLANNING*

O CAMP utiliza o conhecimento da qualidade de conexão para planejar o movimento robótico com o objetivo de melhorar sua performance em uma determinada tarefa, mantendo determinadas restrições de comunicação especificadas pelo problema em questão [8].

Existem algumas maneiras de categorizar esse planejamento, como

- pela modelagem do canal de comunicação: *Path loss* (Perda de percurso), *Shadowing* (Sombreamento) ou *Multipath fading* (Desvanecimento multipercurso) [42];
- pela quantidade de robôs: planejamento individual ou de time;
- pela necessidade de conexão: conexão contínua ou intermitente.

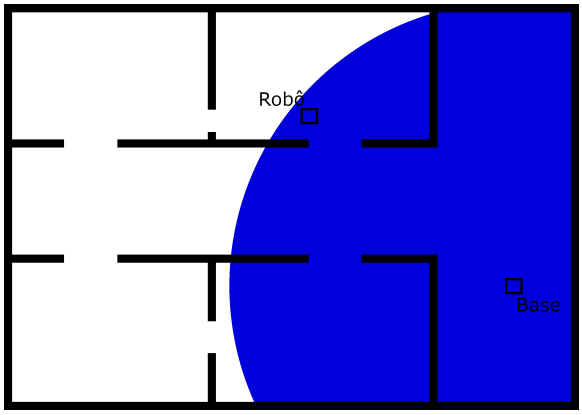
A modelagem do canal de comunicação influencia muito a complexidade do planejamento, sendo que, geralmente, quanto maior a complexidade, mais informação o modelo carrega. O excesso de informação geralmente requer um tempo maior de processamento, gerando resultados mais detalhados e em teoria se adequando melhor ao ambiente apresentado. Com modelos mais simples o processamento é mais rápido, porém várias características do ambiente podem não ser incorporadas no planejamento, criando um modelo não tão bem ajustado ao ambiente. Essa desvantagem de modelos mais simples, entretanto, pode acabar se tornando uma vantagem. Ambientes reais são muito complexos e dinâmicos e um modelo muito complexo que demore a ser processado pode ser muito lento para reagir a essas mudanças.

Os modelos de perda de percurso, geralmente, são os mais simples, eles explicam a perda de força do sinal devido a distância entre emissor e receptor. Essa categoria incorpora, entre outros, os modelos de disco, Figura 2.1a, no qual se assume um raio ao redor do emissor, no qual dentro dele existe comunicação e fora não. E o modelo de raio com desvanecimento, no qual dentro do raio de conexão o sinal vai decaindo com a distância, Figura 2.1b.

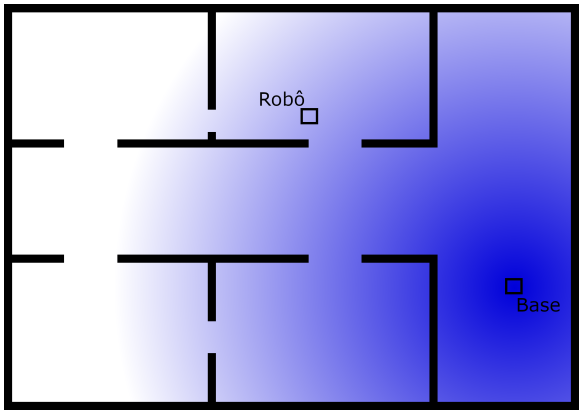
Os modelos de sombreamento levam em conta a presença de obstáculos no ambiente e como eles podem afetar a conexão entre as partes. Esses modelos podem ficar muito complexos, mas a Figura 2.1c mostra uma das versões mais simples possíveis dele, que consideram apenas a visibilidade para determinar a conexão.

Os modelos com desvanecimento multi percurso levam em consideração os múltiplos caminhos que um raio pode fazer devido à reflexão, refração e como isso por reconstruir o sinal em lugares mais distantes. Esses modelos são geralmente mais complexos e na maioria dos casos probabilísticos. A Figura 2.1d mostra um exemplo que assume um modelo de visibilidade onde não há obstáculos e um modelo de desvanecimento multi percurso onde existem obstáculos.

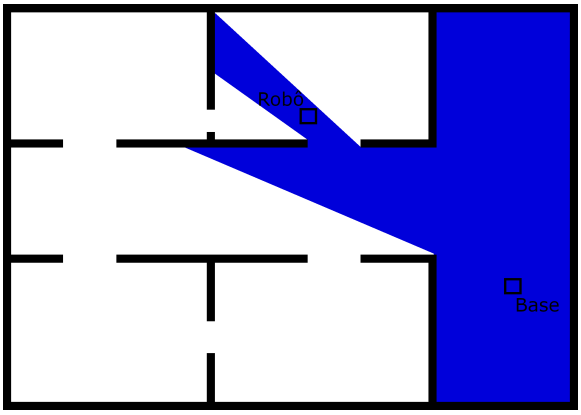
Planejamentos individuais são bem mais simples do que os de time. Existe uma quantidade consideravelmente menor de variáveis envolvidas no problema do que o último. Não existe necessidade de fazer roteamento, existe apenas um robô que se comunica com bases estáticas, no máximo ele terá que decidir para qual base está-



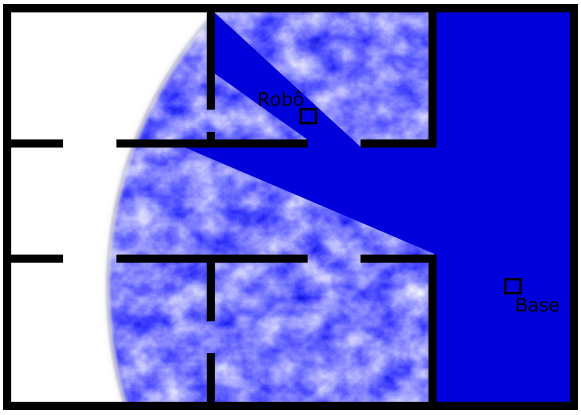
(a) Perda de percurso: Modelo de disco



(b) Perda de percurso: Modelo de raio com desvanecimento



(c) Visibilidade



(d) Visibilidade com ruído

Figura 2.1: Modelos de comunicação

tica ele deve mandar a informação. Geralmente esses planejamentos tentam otimizar alguma métrica, como em [29], [36] e [36]. Não existe muito que o robô possa fazer em relação a conectividade, ou existe conectividade em um determinado ponto, ou não. Dessa forma o espaço de ações possíveis é muito reduzido, o que torna a exploração do espaço de busca mais limitado.

Nos planejamentos de time ([40], [37] e [35]) os robôs podem ser capazes de mudar o perfil de conexão do ambiente. Caso isso seja possível, o sistema se torna bem complexo, tendo que levar em consideração a formação que os robôs se movem, assim como o roteamento que os pacotes irão executar na rede. O espaço de busca é muito mais abrangente que no caso de planejamentos individuais, pois além de ser necessário decidir a ação de múltiplos robôs, é necessário também considerar uma gama maior de possíveis ações para cada um desses robôs.

Planejamentos que possuem necessidades de conexão intermitente, podem contar com restrições mais folgadas do que aqueles que tem necessidade de uma conexão contínua. Problemas como a busca por um determinado objeto em uma área podem contar com uma liberdade de os robôs fazerem uma busca e só voltarem para zonas de conexão de vez em quanto para dar um relatório sobre a situação atual. Já situações de resgate em locais perigosos podem requerer conexões contínuas de forma a haver sempre um monitoramento por parte da equipe de resgate responsável em tempo real.

2.4 ALGORITMOS DE PLANEJAMENTO

Esta subseção tem como objetivo familiarizar o leitor com o tema de algoritmos de planejamento com um enfoque em planejamento baseado em amostras. Caso o leitor deseje obter um maior conhecimento sobre outros tipos de planejamento, como por exemplo o planejamento discreto, o planejamento de movimento combinatorial, planejamento utilizando teoria de decisão, entre outros, recomenda-se a leitura do livro [43], no qual esta seção foi amplamente baseada.

2.4.1 Conceitos básicos

Entre os vários tipos diferentes de conceitos e possíveis significados para a palavra planejamento, os mais relevantes para esse trabalho são os relacionados à robótica. Mais especificamente a necessidade fundamental de transformar especificações em alto nível de tarefas compreendidas por pessoas para tarefas em baixo nível que descrevam os movimentos necessários para realizá-las. Os termos mais comuns para esse tipo planejamento são o planejamento de movimento e o planejamento de trajetória. O planejamento de movimento, de forma geral, busca encontrar um caminho livre de colisões de um ponto inicial para um ponto final. Ele geralmente ignora as restrições de dinâmica e foca principalmente nas translações e rotações necessárias para realizar o movimento. O planejamento de trajetória, geralmente, processa a saída do planejamento de movimento para determinar como o robô irá se mover ao longo da solução encontrada, respeitando suas restrições mecânicas. Para ambos tipos de planejamento, assim como para uma gama bem maior de problemas de planejamento, existem sete conceitos básicos aplicáveis. Esses conceitos são:

Estado O estado é uma representação de uma possível situação do seu problema, por exemplo, o estado pode ser as coordenadas em um plano cartesiano de um robô. O conjunto de todas as possíveis situações que podem ocorrer em um determinado problema de planejamento é chamado de espaço de estados. O

espaço de estados pode ser discreto ou contínuo.

Tempo Tempo pode estar envolvido de forma implícita ou explícita no problema. No primeiro caso, o tempo em que cada ação ocorre não é importante, o que importa é a sequência na qual elas ocorrem. No segundo, a velocidade e quando acontece algo é importante, dessa forma o tempo entra de maneira explícita na formulação do problema. O tempo também pode ser discreto ou contínuo. Problemas com tempo contínuo representam planos que tomam decisões continuamente.

Ações Um plano gera ações que manipulam seu estado. Dessa forma, é necessário que exista uma descrição formal de como as ações podem mudar o estado atual. Duas possíveis formas de representar essas ações são: equações de estado para problemas em tempo discreto; e equações diferenciais para o tempo contínuo. Representar um caminho contínuo por um espaço de estado pode ser uma estratégia muito efetiva para evitar a necessidade de fazer referências explícitas ao tempo. Uma maneira de introduzir incertezas no planejamento é por meio da adição de um agente externo capaz de escolher ações que estejam fora do controle de quem estiver tomando as decisões.

Estados inicial e final Um problema de planejamento geralmente envolve começar em seu estado inicial e buscar alcançar um estado objetivo, ou um entre os estados objetivos. As ações devem ser escolhidas de forma a permitir que isso ocorra.

Critério Esse conceito engloba qual é o objetivo que se deseja alcançar com o planejamento. Os dois objetivos mais comuns são: viabilidade - simplesmente verificar uma maneira de chegar ao objetivo; otimalidade - achar um plano viável que consiga otimizar de alguma maneira um dado critério. Em alguns cenários achar uma solução viável já é um desafio grande o suficiente, apesar de que na robótica, geralmente, se busca a otimalidade de uma solução, em muitos cenários isso é completamente desnecessário e é suficiente achar um solução viável.

Plano Um plano pode especificar uma estratégia ou um comportamento que deve ser executado para realizar um problema. Esse plano pode ser uma sequência de ações a se tomar, ou então algo mais complicado como uma política de ações a se tomar quando algo ocorrer. Isso permite a criação de planos reativos, que consigam reagir ao seu ambiente. Um plano pode ser utilizado de três maneiras: execução - o plano é simplesmente executado em uma simulação ou no mundo real; refinamento - criar um plano melhor, segundo algum critério pré estabelecido; e inclusão hierárquica - ser utilizado como uma ação em um plano maior.

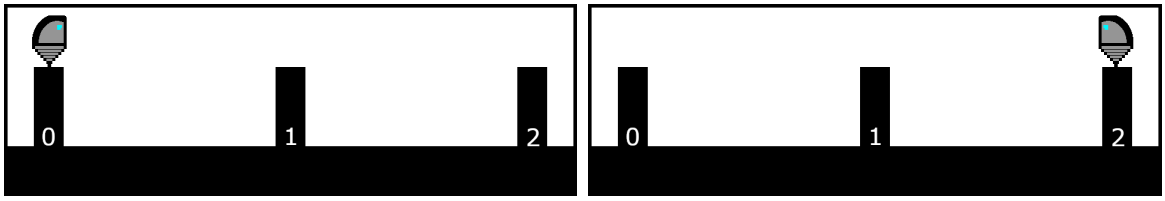
Para entender um pouco mais esses conceitos será analisado o problema de um robô pulador que pode pular entre três postes, representado na Figura 2.2. Para que o robô possa pular de um poste para o outro, ele deve estar de frente para o alvo que ele deseja alcançar.

O estado, neste caso, pode ser representado como o poste sobre o qual ele se encontra e a direção que ele está encarando. O primeiro pode ser representado como um inteiro entre 0 e 2, $P = \{0, 1, 2\}$. A direção pode ser representada como direita ou esquerda, $D = \{e, d\}$. O produto cartesiano desses dois conjuntos forma o nosso espaço de estado,

$$X = P \times D = \{(0, e), (0, d), (1, e), (1, d), (2, e), (2, d)\} \tag{2.1}$$

composto por seis estados. Esse espaço de estados é discreto e finito.

O tempo nesse caso é implícito ao problema, pois o robô precisa apenas pular de um ponto para o outro de maneira sequencial, não é necessário planejar o tempo que o salto leva.



(a) Estado inicial (b) Estado final

Figura 2.2: Exemplo de problema de planejamento

As possíveis ações são pular ou girar. Se o robô em uma posição $p \in P$ escolher girar, seu estado pode mudar de $x = (p, d)$ para $x' = (p, e)$, ou vice e versa. Caso a ação seja pular, sua consequência depende de duas coisas: sua posição atual e sua orientação. Caso ele esteja virado para a direita e sua posição $p \in P$ seja diferente do último poste, ele sairá do estado $x = (p, d)$ para o estado $x' = (p + 1, d)$. Caso ele esteja virado para a esquerda e sua posição $p \in P$ seja diferente do primeiro poste, ele sairá do estado $x = (p, e)$ para o estado $x' = (p - 1, e)$. Caso contrário o robô permanece no mesmo local.

O estado inicial, apresentado na Figura 2.2a, é o estado $x_I = (0, d)$ e o estado final, apresentado na Figura 2.2b, é o estado $x_F = (2, e)$. A otimalidade em relação ao número pulos é um critério aceitável, pois o problema é bem simples e encontrar uma solução ótima não demanda muito trabalho.

Uma maneira de representar um plano ótimo para esse problema é por meio da descrição do plano por meio de todos os estados que compõem o caminho do estado inicial até o final. Um plano ótimo para o problema da Figura 2.2, representado dessa maneira, é $\mathfrak{P} = [(0, d), (1, d), (2, d), (2, e)]$. Outra maneira de presentar um solução ótima seria pela descrição das ações necessárias para chegar no estado final. Por exemplo, $\mathfrak{P} = [\text{Pular}, \text{Pular}, \text{Girar}]$. Durante esse trabalho os planos serão apresentados como o caminho do estado inicial ao final, e não como as ações necessárias para sair de um e chegar no outro.

2.4.2 Planejamento de movimento

O problema conhecido como planejamento de movimento é, fundamentalmente, a obtenção de um caminho sem colisões entre um estado inicial e um estado final para um robô que se move em um ambiente estático e completamente conhecido que pode consistir de um ou mais obstáculos [44]. Planejadores que levam em consideração ambientes dinâmicos também existem [45, 46] e alguns inclusive desconhecem parcialmente o ambiente [47], porém as abordagens clássicas geralmente requerem um conhecimento do ambiente. Apesar dessa restrição, eles são algoritmos muito úteis que geram planos iniciais que podem ser usados como guias para a movimentação do robô que pode reagir a dinamismos do ambiente por meio de mecanismos mais reativos, e ir atualizando o plano mais deliberativo a medida que novas informações são descobertas.

O planejamento de movimento também pode ser visto como um algoritmo que possui um espaço de estados (X) contínuo. As soluções utilizadas nessa classe de algoritmos pode, em sua maioria, ser aplicada na classe mais geral. Esse planejamento ocorre em um ambiente $2D$ ou $3D$ que possui obstáculos, para um robô, partes de um robô ou até mesmo um conjunto de robôs.

A representação do espaço de estados (X) no planejamento de movimento é implícita, incontável e infinita. É possível definir uma transformação entre o ambiente no qual os modelos são definidos e o espaço de estado no qual ocorre o planejamento. Esse espaço de estados (X) no contexto de planejamento de movimento é

mais conhecido como espaço de configuração ($\mathcal{C}$). O espaço de configuração possui uma dimensão elevada e contém uma representação implícita dos obstáculos nele. Após realizada uma busca em $\mathcal{C}$, o caminho contínuo encontrado nele entre a configuração inicial e a final é o plano de movimentação resultante do algoritmo de planejamento.

Uma das principais técnicas em planejamento de movimento é a transformação do modelo contínuo para um discreto. Permitindo que os algoritmos utilizados em planejamentos discretos também se apliquem para o planejamento de movimento. A literatura reconhece duas grandes classes de algoritmos para realizar essa discretização: o planejamento de movimento combinatorial e o planejamento de movimento baseado em amostras. O primeiro é capaz de construir uma representação discreta que representa exatamente o problema original. Esses algoritmos conseguem garantir que existe uma solução viável ou não, entretanto, são algoritmos muito custosos em sua maioria. Isso torna difícil a sua implementação em cenários reais. Por esse motivo, a segunda classe de algoritmos tem tido um enorme uso na última década. O planejamento de movimento baseado em amostra utiliza métodos de detecção de colisão para amostrar o espaço de configuração e realiza uma busca discreta em cima dessas amostras. Esses métodos não conseguem garantir a completude do problema, ou seja, garantir que seja encontrada uma solução viável se ela existir, mas fornecem uma eficiência e facilidade de implementação que conseguem compensar a falta de completude. Eles são recomendados principalmente quando o número de dimensões no espaço de configuração é muito elevado. Por esse motivo, os métodos estudados e utilizados neste trabalho serão métodos baseados em amostragem.

O planejamento de movimento utiliza modelos geométricos para descrever o ambiente e os corpos nesse ambiente. Modelos geométricos podem ser representados pela sua borda, por exemplo uma equação que descreva a circunferência que determina um círculo. Ou pelo seu conteúdo, representação sólida, no qual o círculo é descrito pelo conjunto de todos os pontos que fazem parte dele. O ambiente $\mathcal{W}$ pode ser bidimensional, ou seja $\mathcal{W} = \mathbb{R}^2$, ou tridimensional, $\mathcal{W} = \mathbb{R}^3$. Ele é composto por obstáculos, porções do mundo que estão permanentemente ocupadas, como por exemplo paredes; e robôs que são modelados geometricamente e podem ser controlados por um plano de movimentação. A região de obstáculos $\mathcal{W}_O$ é um subconjunto do ambiente de tal forma que $\mathcal{W}_O$ é o conjunto de todos os pontos em $\mathcal{W}$ que estejam dentro de um ou mais obstáculos, dessa forma $\mathcal{W}_O \subseteq \mathcal{W}$. Sendo $\mathcal{A}$ um dos robôs ele pode ser definido com um subconjunto de $\mathbb{R}^2$ ou $\mathbb{R}^3$.

Um robô $\mathcal{A}$ precisa se mover pelo ambiente $\mathcal{W}$, e para que isso seja possível $\mathcal{A}$ não pode ser um subconjunto de $\mathcal{W}$. Para representar essa movimentação é utilizada uma função chamada de transformação de corpo rígido $h\colon \mathcal{A} \to \mathcal{W}$, que mapeia todos os pontos de $\mathcal{A}$ em $\mathcal{W}$ seguindo duas restrições:

1. a distância entre os pontos de $\mathcal{A}$ precisa ser mantida;
2. e a orientação de $\mathcal{A}$ deve ser mantida.

Dessa forma, para algum ponto $a \in \mathcal{A}$ o ponto $h\,(a)$ é o ponto em $\mathcal{W}$ que é ocupado por a. Assim,

$$h\,(\mathcal{A}) = \{h\,(a) \in \mathcal{W} \mid a \in \mathcal{A}\} \tag{2.2}$$

é a imagem de h que indica todos os pontos em $\mathcal{W}$ ocupados pela transformação do robô.

Nos problemas tratados neste trabalho, os ambientes serão sempre bidimensionais $\mathbb{R}^2$ e os robôs são sempre considerados como pontos adimensionais e sem orientação, sendo assim A contém apenas um ponto $a \in \mathbb{R}^2$. Esse ponto geralmente será o ponto inicial do robô. Levando em consideração a bidimensionalidade e a falta de orientação, a única transformação de corpo rígido possível é a translação.

A translação é uma transformação parametrizada por um vetor de parâmetros $q \in \mathbb{R}^N$ de tal forma que o novo ponto após a translação pode ser obtido por $a' = h(q, a)$. A translação bidimensional de um robô $A = \{a|a = (x, y)\}$ por um parâmetro $q = (x_t, y_t)$ é

$$a' = h(q, a) = (x + x_t, y + y_t). \tag{2.3}$$

2.4.3 O espaço de configuração

Como explicando em 2.4.1, para realizar um planejamento é necessário definir um espaço de estados X. Para o planejamento de movimento ele é conhecido como espaço de configuração $\mathcal{C}$, ou simplesmente espaço $\mathcal{C}$, que é o conjunto das possíveis transformações que podem ser aplicadas ao robô. $\mathcal{C}$ é baseado na mecânica Lagrangiana, leitores que desejem mais detalhes podem começar lendo o capítulo quatro de [43].

Para um robô adimensional em um ambiente bidimensional a única transformação possível é a translação, o conjunto de todas as translação possíveis nada mais é do que $\mathbb{R}^2$, sendo assim $\mathcal{C} = \mathbb{R}^2$. Se considerarmos N robôs adimensionais que podem se mover de maneira independente em um ambiente bidimensional o espaço $\mathcal{C}$ é obtido pelo produto cartesiano das possíveis transformações de cada um dos robôs. Considerando que para um robô $i \in \{1, \ldots, N\}$, $\mathcal{C}_i = \mathbb{R}^2$ seja seu espaço $\mathcal{C}$, o espaço de configuração total desse problema com múltiplos robôs é

$$\mathcal{C} = \mathcal{C}_1 \times \mathcal{C}_2 \times \ldots \times \mathcal{C}_N \tag{2.4}$$
$$\mathcal{C} = \mathbb{R}^2 \times \mathbb{R}^2 \times \ldots \times \mathbb{R}^2 \tag{2.5}$$
$$\mathcal{C} = \mathbb{R}^{2N}. \tag{2.6}$$

Isso demonstra que a complexidade de um problema de planejamento cresce de maneira exponencial com o número de robôs. Outros espaços $\mathcal{C}$ famosos são o $SE(2)$ e o $SE(3)$ que são respectivamente o produto cartesiano do conjunto de translação e rotações bidimensionais e tridimensionais.

Agora que o espaço $\mathcal{C}$ está definido é possível definir a região de obstáculos $\mathcal{W}_\mathcal{O} \subset \mathcal{W}$. Para gerar essa região, basta remover de $\mathcal{C}$ qualquer ponto que gere algum tipo de colisão com o robô. Considerando que $\mathcal{A} \subset \mathcal{W}$ seja um corpo rígido. Sendo $q \in \mathcal{C}$ a configuração de $\mathcal{A}$, na qual $q = (x_t, y_t)$ para $\mathcal{W} = \mathbb{R}^2$. A região de obstáculo, $\mathcal{C}_{obs} \subseteq \mathcal{C}$ é definida como

$$\mathcal{C}_{obs} = \{q \in \mathcal{C} | h(q, \mathcal{A}) \cap \mathcal{W}_O \neq \emptyset\}, \tag{2.7}$$

que é o conjunto de todas as configurações q no qual a transformação do robô intersecta com a região de obstáculo. Para o caso simples em que $\mathcal{C} = \mathbb{R}^2$,

$$\mathcal{C}_{obs} = \{q \in \mathcal{C} | h(q, \mathcal{A}) \cap \mathcal{W}_\mathcal{O} \neq \emptyset\}, \tag{2.8}$$
$$\mathcal{C}_{obs} = \left\{q \in \mathbb{R}^2 | \mathbb{R}^2 \cap \mathcal{W}_O \neq \emptyset\right\}, \tag{2.9}$$
$$\mathcal{C}_{obs} = \{q | q \in \mathcal{W}_O\}, \tag{2.10}$$

ou seja, nesse caso $C_{obs} = \mathcal{W}_O$. Para o caso com N robôs deve se levar em conta não apenas a colisão do robô com $\mathcal{W}_O \subset \mathcal{W}$, mas também a colisão com os outros robôs. Considerando $\mathcal{A}_i$ como o i-ésimo robô

$$\mathcal{C}_{obs} = \left(\bigcup_{i=1}^{N} \{q \in \mathcal{C} | h(q, \mathcal{A}_i) \cap \mathcal{W}_O \neq \emptyset\} \right) \bigcup \left(\bigcup_{i=1}^{N} \bigcup_{j=1, j \neq i}^{N} \{q \in \mathcal{C} | h(q, \mathcal{A}_i) \cap h(q, \mathcal{A}_j)\} \right). \quad (2.11)$$

Sendo os robôs adimensionais e sem orientação isso se reduz a

$$\mathcal{C}_{obs} = \left(\bigcup_{i=1}^{N} \{q \in \mathcal{C} | h(q, \mathcal{A}_i) \cap \mathcal{W}_O \neq \emptyset\} \right) \bigcup \left(\bigcup_{i=1}^{N} \bigcup_{j=1, j \neq i}^{N} \{q \in \mathcal{C} | h(q, \mathcal{A}_i) \cap h(q, \mathcal{A}_j)\} \right), \quad (2.12)$$

$$\mathcal{C}_{obs} = \left(\bigcup_{i=1}^{N} \{q | q \in \mathcal{W}_O\} \right) \bigcup \left(\bigcup_{i=1}^{N} \bigcup_{j=1, j \neq i}^{N} \{q \in \mathcal{C} | h(q, \mathcal{A}_i) \cap h(q, \mathcal{A}_j)\} \right), \quad (2.13)$$

$$\mathcal{C}_{obs} = \{q | q \in \mathcal{W}_O\} \bigcup \left(\bigcup_{i=1}^{N} \bigcup_{j=1, j \neq i}^{N} \{q \in \mathcal{C} | h(q, \mathcal{A}_i) \cap h(q, \mathcal{A}_j)\} \right). \quad (2.14)$$

O espaço que sobra é chamado de espaço livre e é definido como

$$\mathcal{C}_{free} = \mathcal{C} \backslash \mathcal{C}_{obs}. \quad (2.15)$$

Dessa forma, é possível definir formalmente os componentes que formam o problema de planejamento de movimento

1. Um ambiente $\mathcal{W}$.
2. Uma região de obstáculos $\mathcal{W}_{\mathcal{O}} \subset \mathcal{W}$.
3. Um conjunto de robôs $\mathcal{A}$ que possuam funções $h \colon \mathcal{A} \to \mathcal{W}$ que mapeiem os corpos dos robôs ao ambiente.
4. Um espaço de configuração $\mathcal{C}$ que determina o conjunto de todas as possíveis transformações $h \colon \mathcal{A} \to \mathcal{W}$ que podem ser aplicadas aos robôs. Ele é divido entre o espaço livre $\mathcal{C}_{free}$ e o espaço ocupado $\mathcal{C}_{obs}$.
5. Uma configuração inicial $q_i \in \mathcal{C}_{free}$.
6. Uma configuração final $q_f \in \mathcal{C}_{free}$.
7. Um algoritmo que deve calcular um caminho contínuo $\mathfrak{P}$ por $\mathcal{C}_{free}$ de tal forma que seu início seja em q_i e seu final seja em q_f. Ou reportar que esse caminho não existe.

A imagem da Figura 2.3 mostra uma representação abstrata de um espaço $\mathcal{C}$ com seus espaços livres em branco $\mathcal{C}_{free}$ e os espaços ocupados em vinho $\mathcal{C}_{obs}$. Uma possível solução para um problema de planejamento é mostrado com uma linha que conecta q_i e q_f. Esse desenho pode representar qualquer problema desse tipo, desde um simples planejamento bidimensional até múltiplos braços robóticos que devem se movimentar evitando colisão. Todos esses problemas decaem em algo parecido com a Figura 2.3.

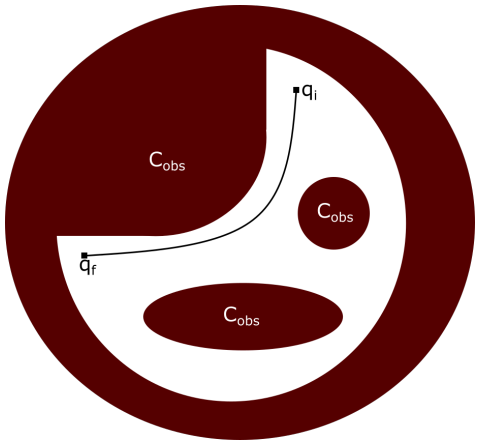


Figura 2.3: Espaço de estados

2.4.4 Planejamento baseado em amostras

O planejamento baseado em amostra surgiu como uma forma de evitar construir de maneira explícita o $\mathcal{C}_{obs}$. Por meio da amostragem, se espera que muitos problemas práticos de alta dimensionalidade, e com muitos graus de liberdade, possam ser resolvidos de maneira eficiente [48]. A principal filosofia que permite o funcionamento desses algoritmos é o detector de colisão, Figura 2.4. Ele é considerado pelo algoritmo como uma caixa preta, fazendo com que o desenvolvimento desses algoritmos possa ocorrer de maneira independente do modelo geométrico associado ao problema. Quem deve se preocupar com isso é o detector de colisão, permitindo que o algoritmo não precise representar de maneira explícita o $\mathcal{C}_{obs}$. Isso permite que problemas com ambientes de alta complexidade geométrica possam ser resolvidos sem precisar levá-la em consideração.

Como afirmado anteriormente, a completude nessa classe de algoritmos é mais fraca do que a de outras. Um algoritmo é completo se e somente se, ele é capaz de achar uma solução, ou declarar que ela não existe, em tempo finito. Um algoritmo de planejamento baseado em amostras não consegue garantir isso, no entanto, caso sua amostragem seja densa, ou seja, caso as amostras se aproximem de qualquer configuração quando as iterações tenderem a infinito, ele é chamado de completo para resolução. Isso significa que, caso exista uma solução, ela será encontrada em tempo finito, porém, caso ela não exista o algoritmo pode nunca terminar sua execução. No caso de amostragem aleatórias, que sejam densas com probabilidade um, essa noção vira a de completude probabilística. Ou seja, se uma solução existe com um número suficiente de amostragens a probabilidade de se encontrar uma solução converge para um.

A grande maioria dos algoritmos baseados em amostras necessitam de uma medida de distância ρ. Essa distância ρ normalmente implica em um espaço métrico. Para o caso $\mathbb{R}^n$ normalmente se utiliza a métrica

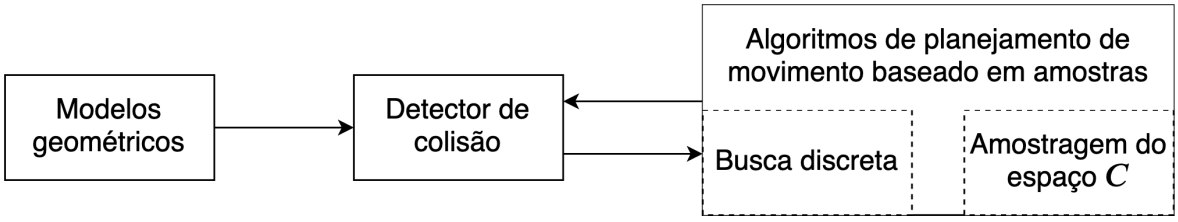


Figura 2.4: A filosofia por trás do planejamento baseado em amostras. Imagem traduzida de [43].

euclidiana. A distância ρ entre duas configurações x e x' é calculada para $p \geq 1$ como

$$\rho\left(x, x'\right) = \left(\sum_{i=1}^{n} \mid x_i - x_i' \mid^p\right)^{\frac{1}{p}}. \tag{2.16}$$

No entanto, muitos espaços $\mathcal{C}$ não são $\mathbb{R}^n$, tornando necessária a definição desses espaços métricos. Os seguintes axiomas devem ser seguidos de forma a transformar o espaço topológico X do espaço $\mathcal{C}$ em um espaço métrico. Considerando o espaço métrico (X, ρ), a sua função $\rho : X \times X \rightarrow \mathbb{R}$ deve cumprir os seguintes requisitos, para qualquer $a, b, c \in X$:

1. Seu valor deve ser não negativo, finito e real: $0 \leq \rho\left(a, b\right) \leq +\infty$.
2. Reflexividade: $\rho\left(a, b\right) = 0$ se e somente se $a = b$.
3. Simetria: $\rho\left(a, b\right) = \rho\left(b, a\right)$
4. Desigualdade triangular: $\rho\left(a, b\right) + \rho\left(b, c\right) \geq \rho\left(a, c\right)$.

Outras partes fundamentais para os algoritmos baseados em amostras são a amostragem e a detecção de colisões. A amostragem em sua maioria é feita de maneira aleatória. Bons algoritmos de amostragem geralmente possuem duas características: baixa dispersão e baixa discrepância. A dispersão pode ser vista como uma medida de espaços sem amostras no seu espaço métrico. Por exemplo, usando medidas euclidianas em um espaço bidimensional seria equivalente a medida do círculo de maior raio que não possui nenhuma amostra. Ter baixa dispersão significa que os espaços sem amostra são pequenos. Uma maneira simples de encarar a baixa discrepância é se assegurar que sua amostragem não forme grades. Grades geram alinhamentos entre os pontos que podem impossibilitar que certas partes do espaço sejam alcançados graças a sua geometria.

A detecção de colisão é geralmente uma parte muito fundamental desses algoritmos. Ela é geralmente a parte que demanda mais poder computacional e por consequência é o que consome mais tempo de execução do algoritmo. [43] fornece muitos detalhes interessantes sobre esse tema. Mais detalhes sobre a detecção de colisão serão fornecidos no capítulo seguinte.

O planejamento baseado em amostras é dividido em dois grandes grupos: os de amostragem e busca incremental e os baseados em *roadmaps*. O primeiro grupo é mais adequado para os casos quando se deseja apenas fazer uma busca no ambiente entre um configuração inicial e uma final. O segundo é mais adequado quando várias buscas diferentes serão realizadas.

Os algoritmos de busca e amostragem incrementais realizam a busca ao mesmo tempo que constroem o espaço por amostragem. Esses algoritmos possuem apenas um ponto inicial e um final, portanto um preprocessamento não traz um ganho de performance. A busca, geralmente, é feita por meio de um grafo topológico não direcional conhecido como grafo de busca $\mathcal{G}\left(V, E\right)$, em que V são os nós e E as arestas desse grafo. Os nós V são configurações $q_a \in \mathcal{C}_{free}$ e as arestas representam caminhos contínuos entre duas configurações $q_a, q_b \in \mathcal{C}_{free}$. A Figura 2.5 mostra um desses grafos após uma típica execução desse tipo de algoritmo. O plano $\mathfrak{P}$ calculado pode ser visualizado em verde.

Utilizando esse grafo de busca, a maioria dos algoritmos de planejamento de movimento baseado em amostras com busca e amostragem incrementais seguem o seguinte modelo:

1. Inicialização: O grafo de busca $\mathcal{G}\left(V, E\right)$ deve ser inicializado com pelo menos um nó em V e nenhuma

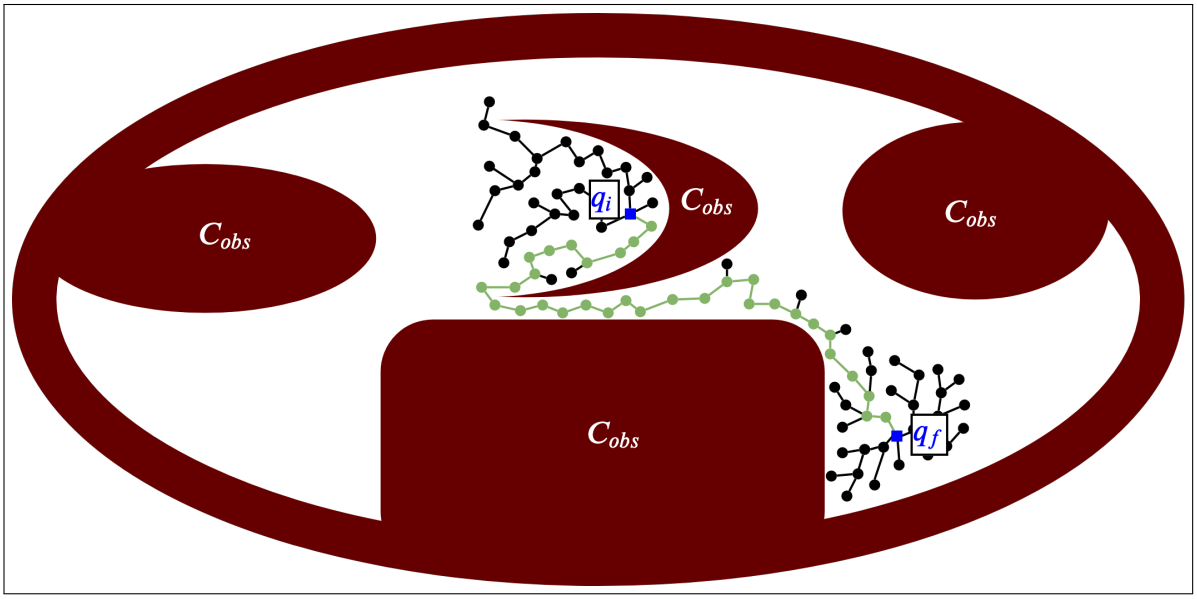


Figura 2.5: Exemplo de um algoritmo de planejamento baseado em amostras com busca e amostragem incrementais.

aresta em E. V pode conter inicialmente qualquer ponto, ou conjunto de pontos, em $\mathcal{C}_{free}$. O mais comum é ele conter q_i , q_f ou ambos.

2. Método de escolha do nó (*Vertex Selection Method - VSM*): Um nó $q_{cur} \in V$ é escolhido para ser expandido.

3. Método de planejamento local (*Local Planning* Method - LPM): Para algum nó novo $q_{new} \in \mathcal{C}_{free}$ que pode ou não ser representado como um nó em V, deve-se tentar construir um caminho contínuo por $\mathcal{C}_{free}$ de tal forma que seu ponto inicial seja q_{cur} e seu ponto final seja q_{new}. Utilizando o detector de colisão esse caminho deve ser averiguado para ver se algum de seus pontos possui uma colisão. Caso esse caminho não esteja livre de colisões é necessário voltar ao segundo passo.

4. Inserir uma nova aresta no grafo: O caminho gerado no passo anterior deve ser inserido em E como uma aresta entre q_{cur} e q_{new}. Caso q_{new} ainda não esteja em V, ele deve ser inserido.

5. Buscar uma solução: Nessa etapa, verifica-se se o grafo $\mathcal{G}(V, E)$ atual possui um caminho solução para o problema.

6. Repetir o passo 2: Enquanto uma solução não tiver sido encontrada ou um ponto de parada não tiver sido alcançado continua-se realizando o algoritmo. Caso um ponto de parada for alcançado sem achar uma solução, o algoritmo deve retornar que falhou.

Os algoritmos com *roadmaps* são utilizados quando se deseja fazer várias buscas no mesmo espaço $\mathcal{C}$. Eles foram inicialmente chamados de *roadmaps* probabilísticos em [49], no entanto eles tem sido mais conhecidos como *roadmaps* baseados em amostras. Eles também são representados por grafos de busca $\mathcal{G}(V, E)$. Um possível *roadmap* para o problema da Figura 2.5 pode ser visualizado na Figura 2.6.

Para otimizar as várias buscas esse tipo de planejamento é dividido em duas partes principais:

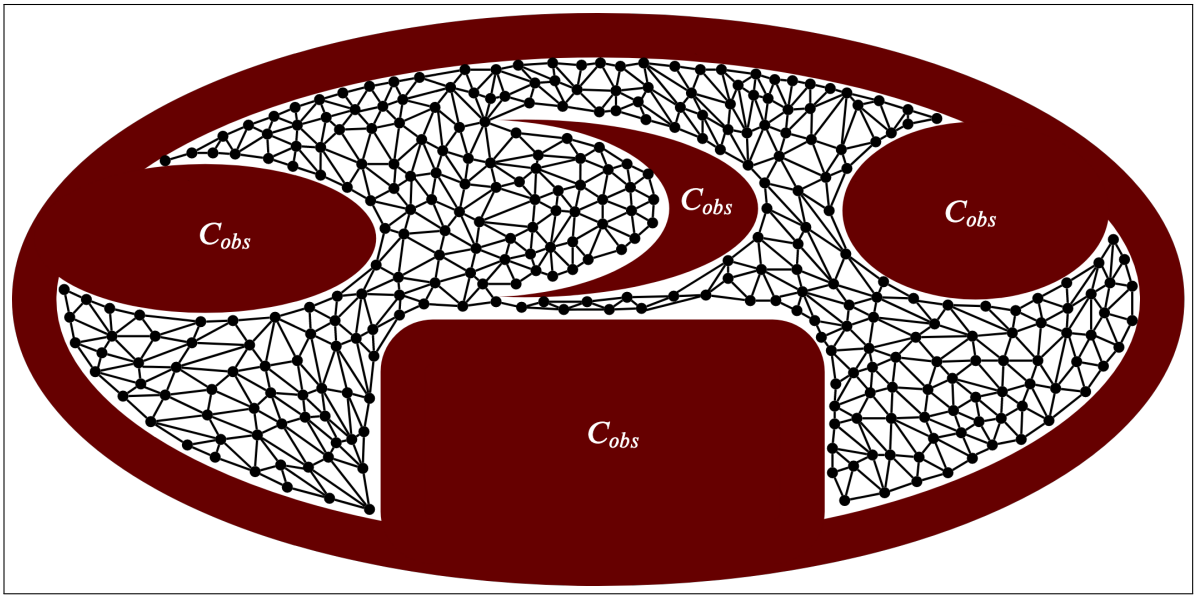


Figura 2.6: Exemplo de um algoritmo de planejamento baseado em amostras com *roadmaps*.

1. Fase de preprocessamento: Essa fase se preocupa em construir $\mathcal{G}(V, E)$ de uma forma que seja possível utilizá-lo para realizar buscas no futuro de maneira eficiente. Essa representação é o que é chamada de *roadmap* e deve ser capaz de cobrir de alguma forma todas as partes de $\mathcal{C}_{free}$ (como representado na Figura 2.6).

2. Fase de busca: Durante essa fase é fornecido um par q_i e q_f, cada uma dessas configurações deve ser conectada ao grafo $\mathcal{G}(V, E)$ por meio de um planejador local. Depois disso uma busca discreta é realizada nesse grafo, utilizando qualquer tipo de algoritmo de busca discreta, de forma a obter uma sequência de arestas que forma um caminho de q_i para q_f.

2.4.5 Buscas em árvores aleatórias

Uma das abordagens de algoritmos de busca e amostragem incrementais que são mais utilizadas e que possuem uma boa performance são as árvores densas de exploração rápida (*rapidly exploring dense trees* - RDTs). Esses algoritmos constroem árvores de busca no espaço $\mathcal{C}_{free}$ que com o passar das suas iterações aumenta a sua resolução até ter preenchido todo o espaço. Quando as amostras utilizadas para a construção das árvores são aleatórias, elas passam a ser conhecidas como árvores aleatórias de exploração rápida (*rapidly exploring random trees* - RRTs).

Duas das principais vantagens das RRTs são sua velocidade e sua simplicidade. O Algoritmo 2.1 mostra uma maneira bem simples de construir uma RRT como a da Figura 2.7.

A função $\alpha : \mathbb{N} \rightarrow \mathcal{C}$ é a função de amostragem, sendo $\alpha(i)$ a i-ésima amostra. $S(\mathcal{G})$ representa todas as possíveis configurações alcançáveis através de $\mathcal{G}$. Isso significa não apenas os nós de V mas também as configurações que compõem os caminhos contínuos representados pelas arestas em E. A função $MAIS\_PRÓXIMO$ calcula a configuração $q_n \in S$ mais próxima de $\alpha(i)$. É importante notar que, para alguns espaços $\mathcal{C}$, calcular a configuração mais próxima em S fica muito custoso e o problema é simplificado para procurar apenas em V.

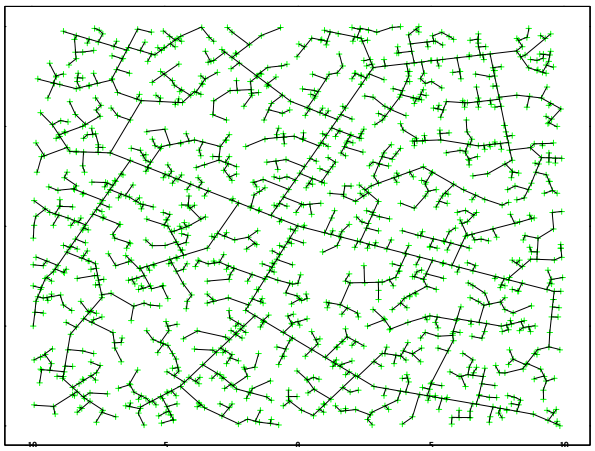


Figura 2.7: RRT após 1000 iterações

Algoritmo 2.1 Simples algoritmo de uma RRT.

1: $\mathcal{G}$.inicializar(q_i)
2: **para** i $\leftarrow$ 1 **até** k **faça**
3: $\quad$ $\mathcal{G}$.adicionar_nó($\alpha(i)$)
4: $\quad$ $q_n \leftarrow$ MAIS_PRÓXIMO($S(\mathcal{G}), \alpha(i)$)
5: $\quad$ $\mathcal{G}$.adicionar_aresta($q_n, \alpha(i)$)
6: **fim para**

A partir da construção dessas árvores de busca é possível classificar os algoritmos de busca pelo número de árvores que eles utilizam:

1. Unidirecional: Os métodos unidirecionais utilizam apenas uma árvore de busca para crescer, geralmente saindo da configuração inicial. Esses métodos são altamente propensos a ficar presos em concavidades e "armadilhas de insetos". "Armadilhas de insetos" são situações como a da Figura 2.8, na qual uma configuração esteja cercada por obstáculos de forma a se encontrar em uma concavidade e só exista uma saída bem estreita na junção de duas concavidades. Nesses casos, quem está dentro da armadilha tem muita dificuldade de sair, mas quem está fora consegue entrar relativamente fácil.

2. Bidirecional: Os métodos bidirecionais criam duas árvores de busca, uma a partir da configuração inicial e outra a partir da configuração final. Isso além de acelerar a busca, ainda ajuda em casos onde uma das duas configurações esteja presa em uma "armadilha de insetos". A busca geralmente opera mudando qual árvore está sendo expandida em um dado momento, além disso, ela alterna entre tentar alcançar a outra árvore e explorar aleatoriamente o espaço. A busca termina quando as duas árvores se encontram.

3. Multidirecional: Algumas vezes ambas as configurações estão em regiões de "armadilhas de inseto", ou algum outro tipo de problema. Nesses casos, pode-se adicionar uma ou mais árvores que comecem em pontos aleatórios do espaço $\mathcal{C}$ para tentar achar meios que contornem essas dificuldades. Mas isso complica bastante as coisas, pois agora várias perguntas podem surgir, como: Quantas árvores usar? Como escolher qual árvore se deve expandir? Em direção a qual das árvores uma delas deve tentar crescer? Entre muitas outras perguntas que tornam o problema difícil de modelar.

Agora que o básico sobre RRTs foi analisado, serão apresentados mais a fundo os algoritmos de RRT

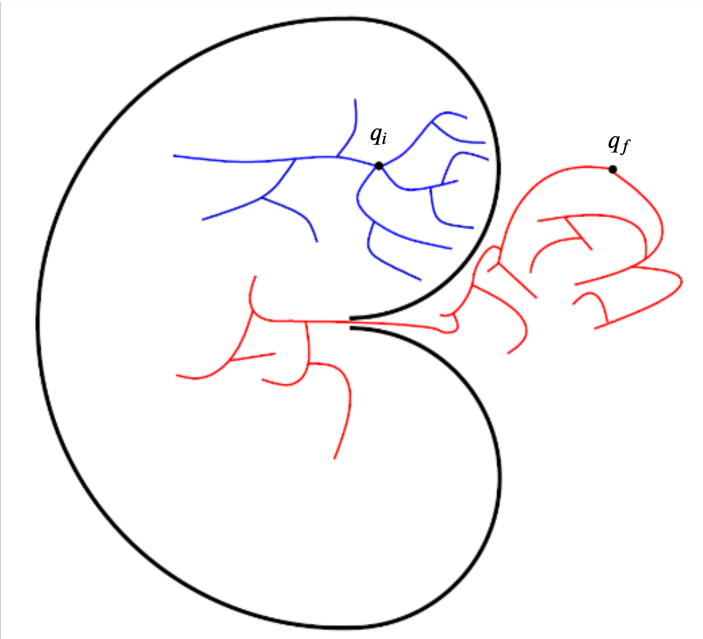


Figura 2.8: “Armadilha de inseto”. Imagem traduzida de [43].

(Algoritmo 2.2) e RRT* (Algoritmo 2.3). Ambos foram adaptados das versões de [50] desses algoritmos. As principais diferenças são que: neste trabalho eles são apresentados como uma função que cresce a árvore em um nó por vez; e a escolha de uma nova configuração pode ser aleatória ou não. Essas escolhas foram feitas para poder agregar ambos os algoritmos em algoritmos de busca mais genérico para esse tipo de árvores, como o algoritmo de busca em uma única árvore (Algoritmo 2.4) e o algoritmo de busca bidirecional balanceado (Algoritmo 2.5).

O RRT foi originalmente proposto em [51] e tem sido extensamente utilizado desde então, dando origem a diversas implementações e algoritmos derivados dele. O Algoritmo 2.2 apresenta a versão utilizada neste trabalho. Essa versão é iterativa, crescendo a árvore uma amostra por vez, devendo ser chamada múltiplas vezes para gerar uma árvore completa. A função descrita nesse algoritmo recebe quatro parâmetros de entrada: o índice da iteração atual k; a amostra inicial q_i, de onde a árvore deve iniciar sua busca; a amostra final q_f, que a árvore deseja alcançar; e o grafo de busca completo que representa a árvore de busca $\mathcal{G}$.

O primeiro passo é determinar de algum modo se nessa iteração a árvore irá crescer em direção ao alvo q_f ou em uma direção aleatória dada pela série aleatória α. A função responsável por isso é a DEVE_ESCOLHER_OBJETIVO, a implementação mais comum desse tipo de função é por meio de uma simples distribuição de Bernoulli com uma probabilidade p de escolher q_f e uma probabilidade $1 - p$ de se escolher uma amostra da série α. Independente da escolha, a amostra candidata é chamada de $q_{candidato}$. Essa etapa corresponde ao passo 2 do modelo de busca incremental apresentado na subseção 2.4.4 chamado de método de escolha do nó.

Com o destino de crescimento definido, é necessário escolher a amostra $q_{mais\_próximo}$ mais próxima de $q_{candidato}$ contida em $\mathcal{G}$, de acordo com a função de distância ρ descrita na subseção 2.4.4. Como o espaço de configuração dos problemas abordados será de uma dimensionalidade grande, somente $\mathcal{G}$ e não $S(\mathcal{G})$ serão considerados nesse algoritmo, diferentemente do Algoritmo 2.1.

Em seguida, a função CONTROLE_DE_AVANÇO determina qual será de fato a nova amostra a ser considerada para análise q_{novo}. Ela verifica se a distância $\rho(q_{mais\_próximo}, q_{candidato})$ é maior que um limiar

Algoritmo 2.2 Crescimento RRT completo.

função CRESCE_RRT(k, $q_i, q_f, \mathcal{G}$)
  se DEVE_ESCOLHER_OBJETIVO(k) **então**
    $q_{candidato} \leftarrow q_f$
  senão
    $q_{candidato} \leftarrow \alpha(k)$
  fim se
  $q_{\text{mais\_próximo}} \leftarrow$ MAIS_PRÓXIMO($\mathcal{G}, q_{candidato}$)
  $q_{novo} \leftarrow$ CONTROLE_DE_AVANÇO($q_{\text{mais\_próximo}}, q_{candidato}$)
  se DETECTA_COLISÃO($q_{\text{mais\_próximo}}, q_{novo}$) **então**
    retorna FALHA
  fim se
  $\mathcal{G}$.adicionar_nó(q_{novo})
  $\mathcal{G}$.adicionar_aresta($q_{\text{mais\_próximo}}, q_{novo}$)
  retorna q_{novo}
fim função

$\lambda \in \mathbb{R}^+$ previamente definido. Caso isso ocorra, uma nova amostra q_{novo} é calculada de forma a obedecer essa restrição. Por exemplo, no caso vetorial essa nova amostra seria obtida da seguinte maneira

$$q_d = q_{candidato} - q_{mais\_próximo}, \tag{2.17}$$

$$q'_d = q_d \frac{\lambda}{\|q_d\|}, \tag{2.18}$$

$$q_{novo} = q'_d + q_{mais\_próximo}. \tag{2.19}$$

Sendo $\|\|$ a norma vetorial, $+$ a soma vetorial e $-$ a subtração vetorial. Isso quer dizer que essa nova amostra deve ser na mesma direção de $q_{candidato}$, porém a uma distância λ de $q_{mais\_próximo}$. Caso a distância seja menor que o limiar λ, q_{novo} será exatamente $q_{candidato}$.

O penúltimo passo é verificar se existe uma conexão contínua e direta entre q_{novo} e $q_{candidato}$ que não esteja completamente dentro de $\mathcal{C}_{free}$. A seção 5.2 de [43] mostra algumas das possíveis técnicas para implementar a função DETECTA_COLISÃO. A técnica escolhida para esta implementação foi a de amostragem do caminho em intervalos mínimos δ. Esse intervalo δ deve ser maior do que a menor dimensão de obstáculo que se deseja considerar. Como esse caminho se limita a um tamanho λ, no pior caso $\frac{\lambda}{\delta}$ amostras serão analisadas para verificar se existe alguma colisão. Essa etapa, juntamente com as duas anteriores, correspondem ao passo 3 do modelo de busca incremental apresentado na subseção 2.4.4 chamado de método de planejamento local.

Por fim, caso uma colisão seja detectada no passo anterior, o algoritmo retorna uma falha em crescer a árvore. Caso contrário, q_{novo} é adicionado ao grafo de busca $\mathcal{G}$, juntamente com uma aresta entre q_{novo} e $q_{mais\_próximo}$. Essa etapa representa o passo 4 do modelo de busca incremental. Os passos 1, 5 e 6 são deixados de lado nesse algoritmo incremental, eles são executados pelos algoritmos de busca 2.4 e 2.5.

O RRT* é uma expansão do algoritmo RRT, ele foi originalmente proposto em [52]. A principal diferença entre os dois é que o o RRT* é assintoticamente ótimo. Isso quer dizer que o custo da solução encontrada pela árvore converge ao custo ótimo com o passar das iterações. O único problema é que, apesar de ambos possuírem a mesma complexidade computacional de $O(n \log n)$ para seu processamento, o RRT* é mais custoso por uma constante. Os algoritmos são bem parecidos, possuindo inclusive alguns dos mesmos passos do Algoritmo 2.2.

Algoritmo 2.3 Crescimento RRT* completo.

função CRESCE_RRT*($k, q_i, q_f, \mathcal{G}, M_C$)
  se DEVE_ESCOLHER_OBJETIVO(k) **então**
    $q_{candidato} \leftarrow q_f$
  senão
    $q_{candidato} \leftarrow \alpha(k)$
  fim se
  $q_{\text{mais\_próximo}} \leftarrow$ MAIS_PRÓXIMO($\mathcal{G}, q_{candidato}$)
  $q_{novo} \leftarrow$ CONTROLE_DE_AVANÇO($q_{\text{mais\_próximo}}, q_{candidato}$)
  se DETECTA_COLISÃO($q_{\text{mais\_próximo}}, q_{novo}$) **então**
    retorna FALHA
  fim se
  $\mathbf{q}_{\text{próximo}} \leftarrow$ PRÓXIMO($\mathcal{G}, q_{novo}$)
  $q_{raiz} \leftarrow$ ESCOLHER_RAIZ($\mathbf{q}_{\text{próximo}}, q_{\text{mais\_próximo}}, q_{novo}, M_C$)
  $\mathcal{G}$.adicionar_nó(q_{novo})
  $\mathcal{G}$.adicionar_aresta($q_{\text{raiz}}, q_{novo}$)
  $\mathcal{G} \leftarrow$ REROTEAR($\mathcal{G}, M_C, \mathbf{q}_{\text{próximo}}, q_{raiz}, q_{novo}$)
  retorna q_{novo}
fim função

Por esse motivo, apenas as diferenças serão explicadas a seguir.

A primeira diferença é que a função iterativa da versão utilizada neste trabalho, apresentada no Algoritmo 2.3, necessita de uma estrutura extra, o mapa de custo M_C. Esse mapa guarda o custo atual de todas as amostras em $\mathcal{G}$ para alcançar a raiz da árvore q_i. Ele será necessário para calcular o custo ótimo da solução.

Os passos iniciais são idênticos aos do Algoritmo 2.2. Após obter com sucesso a nova amostra q_{novo}, ao invés de adicionar ela ao grafo de busca, novas etapas são executadas. Inicialmente são obtidas todas as amostras $\mathbf{q}_{\text{próximo}}$ pertencentes à árvore de busca próximas à amostra q_{novo}. Quantas amostras e qual a distância depende do algoritmo escolhido para a função PRÓXIMO. O algoritmo escolhido para essa implementação é o mesmo da implementação original, na qual o raio de escolha das amostras próximas diminui com o aumento da árvore segundo a fórmula

$$r_{próximo} \quad = \quad \gamma \sqrt[d]{\frac{\log |V|}{|V|}}. \tag{2.20}$$

Sendo que γ é um parâmetro do algoritmo, $|V|$ é o número de amostras na árvore e d é a dimensão do espaço $\mathcal{C}$. Essa função decai de forma que o número médio de amostras na vizinhança decaia proporcionalmente a $\log |V|$ [52].

Agora que a vizinhança $\mathbf{q}_{\text{próximo}}$ está definida, o algoritmo procura nessa região se existe algum candidato cuja função de custo total seja menor do que a de $q_{mais\_próximo}$. A função de custo total é definida como o custo da amostra q_{novo} até uma uma amostra $q_a \in \mathbf{q}_{\text{próximo}}$ somado ao custo salvo no mapa de custo M_C de q_a. A amostra q_a com menor custo total ou $q_{mais\_próximo}$, caso seu custo total seja menor, será escolhida para ser a conexão q_{raiz} de q_{novo} a árvore. Com q_{raiz} definido, q_{novo} é inserido a V e a aresta entre q_{raiz} e q_{novo} é inserida a E. É importante notar que caso o caminho entre q_a e q_{novo} passe por um obstáculo seu custo é

considerado infinito.

Por fim, um novo passo é executado pela função REROTEAR. Ela analisa todas as amostras $q_a \in \mathbf{q}_{\text{próximo}}$ e verifica se o custo total de conectar q_a a q_{novo} é menor do que o custo total atual dessa amostra ser conectada à amostra raiz atual. Caso isso aconteça, q_a é desconectado de sua raiz atual e reconectado a q_{novo}. Dessa maneira, amostras antigas podem ser atualizadas para se manterem sempre com um custo mínimo para alcançar a raiz da árvore q_i.

Esses dois algoritmos iterativos de crescimento de árvores aleatórias de busca permitem que elas sejam tratadas de maneira similar. A única diferença é que o RRT* exige a estrutura extra do mapa de custo M_C. Sendo assim, se considerarmos que a árvore de busca T seja representada por $\mathtt{T} = \mathcal{G} = (V, E)$ para o caso do RRT e como $\mathtt{T} = (\mathcal{G}, M_C) = (V, E, M_C)$ no caso do RRT*, dessa forma incorporando M_C, ambos os Algoritmos 2.4 e 2.5 de busca em árvore podem utilizar tanto o Algoritmo 2.2 como o Algoritmo 2.3 para o passo de crescimento.

Ambos os algoritmos de busca em árvore foram extraídos e adaptados de [43]. Para entendê-los melhor é necessário definir a estrutura do objeto T que representa as árvores de buscas. Ela possui alguns métodos associados a ela. O primeiro é o método *inicializar*, ele recebe dois parâmetros, a amostra inicial da árvore e o objetivo de busca da árvore. A amostra inicial é inserida no conjunto V e seu custo é indicado como zero no mapa de custo M_C caso a árvore seja uma RRT*. O objetivo de busca da árvore é apenas salvo como referência para os outros métodos. O método *definir_taxa_de_busca_ao_objetivo* define uma probabilidade p para àquela árvore utilizar no seu passo de crescimento na função DEVE_ESCOLHER_OBJETIVO. O método *definir_objetivo* muda, para aquela árvore, qual é amostra objetivo de busca atual. O método *crescer* é um dos dois algoritmos iterativos de crescimento de árvore apresentados (Algoritmos 2.2 e 2.3). Ele passa todos os parâmetros necessários para essas funções de acordo com o tipo de árvore utilizado. Por fim, o método *caminho_até_raiz* calcula o caminho de uma amostra $q_a \in V$ até a amostra inicial q_i. Ele sobe a hierarquia da árvore a partir de q_a até chegar na raiz principal q_i, todas as amostra que conectam elas formam o caminho que representa o plano de movimento.

Os algoritmos de busca apresentados foram baseados em [43]. O primeiro, Algoritmo 2.4, busca em uma única árvore, é bem simples. Ele começa inicializando a árvore T com a amostra inicial q_i e o alvo de busca q_f. Após isso, ele irá tentar crescer a árvore iterativamente. Ele tenta um máximo de MÁX_TENTATIVAS vezes encontrar q_f, caso isso não ocorra, o algoritmo retorna falha. Para verificar se q_f foi encontrado, ele,

Algoritmo 2.4 Busca em apenas uma árvore.

```
1:  função BUSCA_EM_ÁRVORE(q_i, q_f, LIMIAR)
2:      T.inicializar(q_i, q_f)
3:      para k ← 0 até MÁX_TENTATIVAS faça
4:          q_novo ← T.crescer(k, q_i, q_f)
5:          se q_novo ≠ FALHA então
6:              se ρ(q_novo, q_f) < LIMIAR então
7:                  retorna T.caminho_até_raiz(q_novo)
8:              fim se
9:          fim se
10:     fim para
11:     retorna FALHA
12: fim função
```

Algoritmo 2.5 Busca bidirecional balanceada.

função BUSCA_EM_ÁRVORE_BIDIRECIONAL(q_i, q_f, LIMIAR)
    $\mathtt{T}_a$.*inicializar*(q_i, q_f)
    $\mathtt{T}_b$.*inicializar*(q_f, q_i)
    árvores_se_encontraram ← FALSO
    $k \leftarrow 0$
    enquanto árvores_se_encontraram ≠ VERDADEIRO **faça**
        $\mathtt{T}_a$.*definir_taxa_de_busca_ao_objetivo*(0)
        $\mathtt{T}_b$.*definir_taxa_de_busca_ao_objetivo*(1)
        $q_{a_{novo}} \leftarrow \mathtt{T}_a$.*crescer*($k, q_i, q_f$)
        se $q_{a_{novo}} \neq$ FALHA **então**
            $\mathtt{T}_b$.*definir_objetivo*($q_{a_{novo}}$)
            $q_{b_{novo}} \leftarrow \mathtt{T}_b$.*crescer*($k, q_f, q_{a_{novo}}$)
            se $\rho\left(q_{b_{novo}}, q_{a_{novo}}\right) <$ LIMIAR **então**
                $\mathfrak{P}_a \leftarrow \mathtt{T}_a$.*caminho_até_raiz*($q_{a_{novo}}$)
                $\mathfrak{P}_b \leftarrow \mathtt{T}_b$.*caminho_até_raiz*($q_{b_{novo}}$)
                árvores_se_encontraram ← VERDADEIRO
            fim se
        fim se
        se $|\mathtt{T}_a| > |\mathtt{T}_b|$ **então**
            TROCA($\mathtt{T}_a, \mathtt{T}_b$)
        fim se
        $k \leftarrow k + 1$
        se $k >$ MÁX_TENTATIVAS **então**
            retorna FALHA
        fim se
    fim enquanto
    $\mathfrak{P} \leftarrow$ UNIR($\mathfrak{P}_a, \mathfrak{P}_b$)
    retorna $\mathfrak{P}$
fim função

primeiramente, obtém uma amostra q_{novo}, representando a última amostra adicionada à árvore T. Essa amostra é obtida do algoritmo de crescimento de árvore descrito anteriormente. Se esse algoritmo retornar uma falha, essa etapa é ignorada e a próxima iteração é iniciada. Se a amostra q_{novo} for obtida com sucesso, ele verifica se q_{novo} está a uma distância de q_f menor do que LIMIAR. Sendo isso verdade, a árvore encontrou seu objetivo e o algoritmo retorna o caminho de q_{novo} até q_i pelo método *caminho_até_raiz* descrito anteriormente. Se não, ele continuar procurando.

A busca bidirecional (Algoritmo 2.5) já é mais complexa. Ela inicializa duas árvore $\mathtt{T}_a$ e $\mathtt{T}_b$. A árvore $\mathtt{T}_a$ é inicializada como no caso com uma árvore apenas, com a amostra inicial sendo q_i e o alvo de busca sendo q_f. A árvore T_b faz o processo inverso, ela busca o início a partir do alvo, dessa forma q_f é a amostra inicial e o alvo é q_i. Com isso, o algoritmo irá tentar buscar um caminho entre q_i e q_f. Esse caminho será formado quando as duas árvores se encontrarem. O algoritmo usa a variável "árvores_se_encontraram" para determinar se isso ocorreu. Ele busca esse encontro por meio de um laço que executa até que essa variável seja verdade.

Essa busca no laço inicia definindo a taxa de busca de cada uma das árvores. A árvore $\mathtt{T}_a$ é ajustada

para ter uma probabilidade $p = 0$ de ir em direção ao alvo, enquanto a árvore $\mathtt{T}_b$ tem uma probabilidade $p = 1$ de ir em busca do alvo. Dessa maneira, a primeira procura sempre de maneira aleatória, explorando o ambiente, enquanto a segunda sempre tenta alcançar o seu objetivo. Isso permite combinar exploração, para escapar de mínimos locais, com uma estratégia gulosa de ir em direção ao objetivo. Com isso, ele tenta crescer a árvore $\mathtt{T}_a$ e obter a nova amostra $q_{a_{novo}}$ dessa árvore. Ele define essa amostra como o novo objetivo da árvore $\mathtt{T}_b$ e tenta obter uma nova amostra $q_{b_{novo}}$ por meio do seu crescimento. Para averiguar se as árvores se encontraram, o algoritmo verifica se a distância entre $q_{a_{novo}}$ e $q_{b_{novo}}$ é menor que um dado LIMIAR. Caso isso ocorra, o caminho de cada uma dessas amostras para suas respectivas raízes é salvo em $\mathfrak{P}_a$ e $\mathfrak{P}_b$ e a variável árvores_se_encontram é definida como verdadeiro. Ao final do laço, ele verifica se o número de amostras em $\mathtt{T}_a$ é maior que o número de amostras em $\mathtt{T}_b$, essas árvore trocam de lugar, com a árvore $\mathtt{T}_b$ se tornado a árvore $\mathtt{T}_a$ e vice e versa. Esse passo garante que caso uma das árvores se prenda em um mínimo local, o algoritmo se encarrega de tornar a exploração daquela árvore aleatória. Esse algoritmo tenta uma máximo de MÁX_TENTATIVAS vezes antes de retornar uma falha. Caso o caminho tenha sido achado com sucesso, ele unifica os dois caminhos em $\mathfrak{P}$ por meio da função UNIR gerando um plano de movimento final.

Com isso, os principais algoritmos necessários para compreender as propostas do capítulo seguinte foram apresentados. As próximas seções apresentarão alguns algoritmos adicionais utilizados adiante: o algoritmo de Dijkstra e a triangulação de Delaunay. Esses dois algoritmos são utilizados como auxiliares para, respectivamente, encontrar caminhos de menor custo e conectar amostras de uma nuvem de pontos.

2.4.6 Algoritmo Dijkstra

O algoritmo de Dijkstra [53] foi proposto inicialmente em 1959. Ele soluciona o problema de encontrar o caminho mais curto entre um, ou mais vértices, a todos os outros vértices de um grafo. Dado um grafo $\mathcal{G} = (V, E)$ o algoritmo é capaz de encontrar essas distâncias em uma complexidade temporal de $O(|E| + |V| \log |V|)$ no pior caso. É importante notar que esse algoritmo só funciona se os pesos das arestas E forem não negativos.

Geralmente o algoritmo de Dijkstra é utilizado para calcular a menor distância entre dois dos vértices de V. Porém, sua implementação mais padrão permite que seja calculada a distância de menor custo entre um vértice $s \in V$ para todos os outros vértices de V. A versão utilizada neste trabalho deseja calcular a distância de um conjunto $\mathcal{S} \subset V$, ao invés de apenas um vértice.

O Algoritmo 2.6 descreve uma das implementações mais comuns desse algoritmo. Dado um grafo $\mathcal{G}(V, E)$ e um subconjunto de amostras $\mathcal{S} \subset V$, ele calcula a menor distância de qualquer vértice $u \in V$ para esse subconjunto S. O primeiro passo para isso é definir para todos os vértices $v \in V$ o custo inicial deles no mapa de custos $\mathcal{D}$. Os vértices que pertencerem ao subconjunto $\mathcal{S}$ são inicializados com o valor zero e os que não pertencem são inicializados com o valor infinito.

Com isso, uma *heap* mínima Q é construída a partir de V baseada nos valores do mapa de custo $\mathcal{D}$. Enquanto essa *heap* não estiver vazia, o algoritmo remove o vértice de menor custo u e verifica para cada vértice v adjacente se existe um custo menor possível para ele. Isso é feito comparando o custo atual, $\mathcal{D}(v)$, com o custo do vértice u, $\mathcal{D}(u)$, somado com o peso da aresta entre eles. Esse peso, nos problemas de planejamento de movimento deste trabalho, é dado pela função de distância $\rho(u, v)$. Caso essa soma seja menor que o custo atual $\mathcal{D}(v)$, ele será atualizado para a soma $\mathcal{D}(u) + \rho(u, v)$.

Existem diversas versões mais otimizadas desse algoritmo como por exemplo o A* [54] de 1968. Esse

Algoritmo 2.6 Algoritmo de Dijkstra.

```
1: função DIJKSTRA(G(V, E), S)
2:     para todo v ∈ V
3:         se v ∈ S então
4:             D[v] ← 0
5:         senão
6:             D[v] ← ∞
7:         fim se
8:     fim para
9:     Q ← CONSTRUIR_HEAP(V, D)
10:    enquanto Q ≠ faça
11:        u ← EXTRAIR_MENOR(Q)
12:        para todo v adjacente a u
13:            se D[v] > D[u] + ρ(u, v) então
14:                D[v] ← D[u] + ρ(u, v)
15:            fim se
16:        fim para
17:    fim enquanto
18:    retorna D
19: fim função
```

algoritmo possui uma complexidade temporal bem melhor do que o algoritmo de Dijkstra caso o objetivo seja encontrar o melhor caminho entre dois vértices de um grafo. Isso é obtido por meio de uma função heurística que guia a busca. Vários outros algoritmos otimizam outros aspectos dessa busca. Mas nenhuma delas acabaram se provando úteis para a aplicação desenvolvida neste trabalho, já que será necessário calcular a distância entre todos os pontos do grafo até um subconjunto definido de vértices dentro do mesmo. Nesta aplicação esse mapa de custo $\mathcal{D}$ será calculado uma vez e utilizado múltiplas vezes, compensando um elevado custo computacional inicial com uma simples consulta a um mapa em iterações futuras.

2.4.7 Triangulação de Delaunay e Formas-α

Dado um conjunto de pontos V, uma triangulação fornece uma grafo planar de linhas retas $\mathcal{G}(V, E)$ no qual nenhum par de arestas $e_1, e_2 \in E$ se intersectam propriamente. Isso quer dizer que, elas não se intersectam em nenhum outro ponto além de suas extremidades. Uma triangulação muita famosa e muito utilizada é a triangulação de Delaunay (Figura 2.9). Ela possui uma complexidade temporal de $O(|V| \log |V|)$ e possui a propriedade que o círculo circunscrito em qualquer triângulo formado por arestas de E não possui nenhum ponto de V em seu interior [55].

Essa triangulação foi inventada em 1934 [56] e, além de não possuir pontos no interior de seus triângulos, ela também maximiza o menor ângulo de todos os triângulos dessa triangulação. Dessa forma, se evita a construção de triângulos muito fechados. Um dos fatores contribuintes para essa triangulação é a relação que ela possui com o diagrama de Voronoi [57]. Ao se ligar os centros dos círculos que circunscrevem os triângulos gerados, se obtém o diagrama de Voronoi. Esse diagrama divide o espaço, no qual os pontos triangulados $v \in V$ se encontram, em células. Essas células, chamadas de células de Voronoi, possuem todos os pontos do espaço

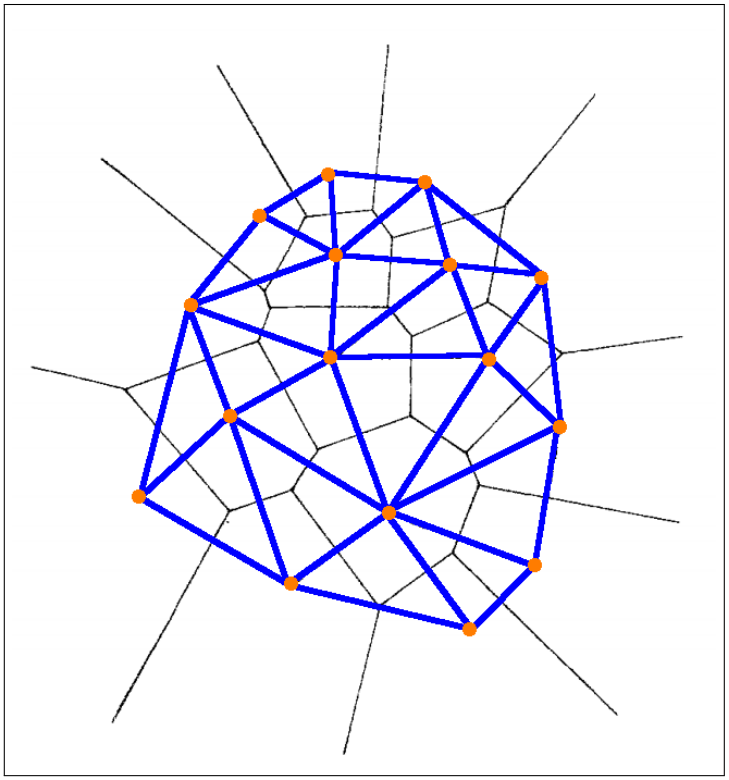


Figura 2.9: Triangulação de Delaunay (em azul), e seu Diagrama de Voronoi (em preto) correspondente, de um conjunto de pontos (em laranja). Adaptado de [55].

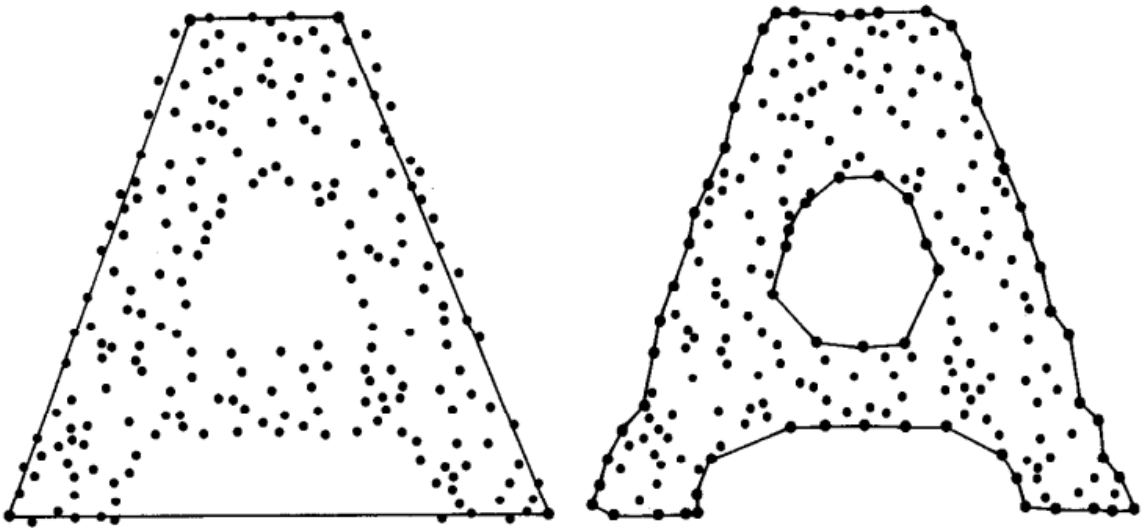
$\mathbb{R}^n$, caso seja um espaço euclidiano, que estão mais próximos do ponto $v \in V$ pertencente àquela célula. A Figura 2.9 mostra um exemplo de triangulação de Delaunay, em azul, de um conjunto de pontos, em laranja. Além disso, a figura também mostra o diagrama de Voronoi desse conjunto de pontos.

Unindo o fato de qualquer círculo circunscrevendo um triângulo não possui qualquer outro ponto em seu interior, que a triangulação maximiza o menor ângulo e seu relacionamento com o diagrama de Voronoi, além de uma série de outras características interessantes, essa triangulação é uma boa candidata para utilizar na modelagem poligonal de uma nuvem de pontos.

Uma outra aplicação dessa triangulação é para a determinação de formas-α (Figura 2.10) [58]. Essa família de formas é um conjunto de segmentos de retas no espaço euclidiano associados a forma de um conjunto finito de pontos. As formas-α são derivadas dos cascos-α, que por sua vez são uma generalização do conceito de cascos convexos.

Um casco-α de um conjunto de pontos V é definido como a interseção de todos os discos de raio $\frac{1}{\alpha}$ que contém todos os pontos de V. Sendo que para isso, o conceito de um disco de raio $\frac{1}{\alpha}$ é generalizado da seguinte maneira: caso $\alpha = 0$, esse disco é um semiplano fechado, e o casco-α se reduz a casco convexo; caso $\alpha > 0$, ele é realmente um disco de raio $\frac{1}{\alpha}$; e caso $\alpha < 0$, ele é o complemento do disco de raio $-\frac{1}{\alpha}$.

Considerando um ponto $p \in V$, ele é considerado um ponto α-extremo de V, se existe um disco de raio $\frac{1}{\alpha}$ generalizado, de tal forma que p esteja na sua borda e esse disco contenha todos os pontos de V. Se existirem dois pontos p e q que sejam α-extremos, e se existe um disco de raio $\frac{1}{\alpha}$ generalizado que contenha ambos esses pontos em sua borda e que contenha todos os outros pontos de V, então p e q são vizinhos-α.



(a) Forma-α positiva

(b) Forma-α negativa

Figura 2.10: Formas-α de um conjunto de pontos. Fonte [58].

Dessa forma, uma forma-α é definida como: dado um número real arbitrário α, e um conjunto de pontos V, a sua forma-α é o grafo em linha reta cujos vértices são os pontos α-extremos e cujas arestas são as conexões entre os vizinhos-α [58]. A Figura 2.10a exibe uma forma-α positiva, e a Figura 2.10b mostra uma forma-α negativa.

As formas-α serão utilizadas neste trabalho para determinar a borda de regiões definidas por uma nuvem de pontos. Além das características já citadas, esse tipo de triangulação foi escolhido por possuir um fundação matemática sólida e por ser fácil de computar [59]. Maneiras de calcular formas-α de dimensionalidade maior foram propostas em [60, 61].

3 PROPOSTA

3.1 INTRODUÇÃO

O capítulo anterior apresentou todos os conceitos básicos necessários para compreender o conteúdo deste capítulo. Recomenda-se fortemente a leitura dele, caso o leitor não esteja familiarizado com os conceitos de: *Communication-Aware Motion Planning* (CAMP) (Seção 2.3); planejamento de movimento baseado em amostras (Subseção 2.4.4); triangulação de Delaunay e Formas-α (Subseção 2.4.7). Além desses conceitos, o leitor deve se familiarizar com alguns algoritmos como: os algoritmos de busca em árvores aleatórias, RRT e RRT* (Subseção 2.4.5) e o algoritmo de Dijkstra (2.4.6).

Como explicado anteriormente, o cenário prático que se deseja implementar é a criação de um sistema de assistentes laboratoriais para o Laboratório de Automação e Robótica – LARA – da Universidade de Brasília – UnB. Ele será executado por um conjunto de três robôs Pioneers que existem no laboratório. Esses robôs vão receber ordens de um servidor e vão permitir também que seja feito um monitoramento em tempo real de suas atividades. Graças a esses requisitos, os robôs não podem perder conexão com a internet em nenhum momento, criando a necessidade de se fazer um CAMP. Infelizmente não houve tempo hábil para que essa implementação de fato fosse feita nos robôs do laboratório.

Esse capítulo será dividido em quatro outras partes. A Seção 3.2 apresentará com mais detalhes esse cenário prático, juntamente com uma modelagem que pode ser aplicada não apenas nesse caso, mas sim de uma forma mais geral. Ela também apresentará de forma mais ampla a solução completa que foi pensada para esse problema e quais partes dessa solução foram de fato implementadas neste trabalho. As três seções seguintes explicam com mais detalhes as soluções implementadas. A primeira delas, Seção 3.3 apresenta as tarefas-α, conceito fundamental introduzido neste trabalho para determinar pontos de apoio necessário para manter a conexão e também definir quantos robôs são necessários para realizar uma determinada tarefa. A Seção 3.4 utiliza as tarefas-α para determinar um custo aproximado de realização de uma tarefa, para que possa ser escolhido qual tarefa deve ser realizada primeiro. Dado que se escolheu a tarefa a se realizar, a Seção 3.5 planeja uma maneira de se realizar a mesma sem perda de conexão.

3.2 DETALHAMENTO E MODELAGEM DO SISTEMA

O sistema de assistentes Pioneers deve ser capaz de navegar pelo primeiro andar do prédio do SG-11. Eles devem ser capazes de realizar tarefas que geralmente podem ser simplificadas como o transporte de objetos entre dois locais. Como por exemplo, buscar um café para um professor, pegar uma impressão e levar para um aluno, buscar algum material no almoxarifado, entre outras coisas. Pessoas relacionadas ao laboratório devem poder entrar a qualquer momento em uma plataforma e adicionar um nova tarefa para ser realizada pelo sistema. Administradores, ou pessoas autorizadas, podem visualizar as câmeras e se desejarem, podem até controlar o robô a distância. Como dito anteriormente, isso gera a necessidade de se manter uma conexão contínua entre qualquer um dos robôs e alguma zona que lhe conceda acesso a Internet.

É possível defender que no cenário apresentado, uma conexão contínua não seja estritamente necessária.

Dessa forma, bastaria garantir que existe uma conexão intermitente. Entretanto, esse cenário tem como objetivo simular situações mais extremas, como as de resgates na qual a conexão intermitente é completamente inadmissível. Por esse motivo é considerado que a conexão do sistema de assistentes deve ser contínua.

O ambiente possui um rede própria para propagação de sinal sem fio, porém essa rede possui um alcance máximo. Os robôs dispõem de hardware capaz de propagar essa conexão, podendo servir como expansores de rede, sendo que um robô pode expandir a rede adquirida de outro robô, formando uma espécie de corrente propagadora do sinal. Dessa forma, o sistema deve ser capaz de decidir se existe uma maneira do robô resolver uma tarefa sozinho, ou se ele vai precisar da ajuda de outros. Caso seja necessária a colaboração, o sistema deve tanto decidir se é possível realizar a tarefa com o número de robôs disponíveis, como planejar uma maneira de movimentar os robôs para realizar a tarefa sem perda de conexão.

Pensando de uma maneira mais geral, o problema pode ser definido da seguinte maneira: um grupo de robôs previamente conhecidos devem estar aptos a realizar tarefas em um ambiente parcialmente conhecido. Dessa forma, o sistema deve estar apto a se adaptar a possíveis mudanças no ambiente, mas também possuindo informações para fazer uma estimativa inicial de suas ações. Com o passar do tempo o sistema vai obtendo mais informações do ambiente e vai adaptando seus modelos de acordo. Deve-se entender aqui que o ambiente é composto por obstáculos tanto físicos, como paredes, cadeiras e mesas, como de conexão, ou seja, um modelo de conexão do ambiente.

Caso uma tarefa esteja fora da área de conexão, os robôs devem ser capazes de agir de maneira colaborativa para realizá-la. Um robô só pode servir como um expansor de rede se ele estiver dentro de uma área de conexão, ou se ele estiver conectado a outro robô que possuir conexão. Para propósitos de planejamento é considerado que dois robôs possuem uma conexão caso exista visada entre eles. Esse modelo de visada foi escolhido, pois como os robôs estão constantemente se locomovendo dentro de um ambiente complexo, é muito difícil modelar os canais de propagação de maneira acurada. Esse modelo ainda apresenta vantagens geométricas que serão mencionadas posteriormente. Com o problema definido, serão apresentadas, agora, algumas modelagens matemáticas necessárias para entender como o problema será resolvido.

Um grupo de robôs, definido por um conjunto $\mathbf{R}$, possui $n_r \in \mathbb{N}$ robôs $r_i \in \mathbf{R}$, com $i \in \{1, \ldots, n_r\}$. Um robô $r_i = \left(r_\tau^i, r_p^i\right)$ é definido por duas características, seu tipo $r_\tau^i \in \mathcal{T}_R$, e sua posição $r_p^i \in \mathcal{W}$. O conjunto $\mathcal{T}_R$ de tipos de robôs deve ser previamente definido, já o conjunto $\mathcal{W}$ representa o ambiente no qual o planejamento irá ocorrer.

Até agora o conceito de tarefas utilizado foi bem amplo. A partir desse momento, o termo tarefa será sempre associado a realização de uma ação por um determinado tempo em uma região específica do ambiente. Dessa forma, alguns dos exemplos que o sistema precisa realizar, como pegar um café para o professor, são impossíveis de realizar em uma única tarefa, já que primeiro o robô deve buscar o café e depois levar até o professor, configurando duas tarefas. Essa ação como um todo será, a partir de agora, chamada de missão. O conjunto de missões $\mathbf{M}$ representa todas as $n_m \in \mathbb{N}$ solicitações recebidas até o momento pelo sistema. Cada missão $m_j \in \mathbf{M}$, com $j \in \{1, \ldots, n_m\}$, é descrita por $m_j = \left(m_{\mathbf{T}}^j, m_\pi^j, m_{\mathfrak{d}}^j\right)$. $m_{\mathbf{T}}^j \subset \mathbf{T}$ é um subconjunto do conjunto de todas as tarefas $\mathbf{T}$ atualmente no sistema. $m_\pi^j \in \Pi$ é a prioridade associada àquela missão, o conjunto Π define quais são os níveis de prioridade existentes no sistema. $m_{\mathfrak{d}}^j \in \mathbb{R}^+$ é o prazo máximo para se realizar essa missão.

Uma tarefa $t_k \in \mathbf{T}$, com $k \in \{1, \ldots, n_t\}$, em que $n_t \in \mathbb{N}$ é o número de tarefas no conjunto $\mathbf{T}$, é definida, também, por três características $t_k = \left(t_{\mathbf{T}}^k, t_p^k, t_\tau^k\right)$, um subconjunto $t_{\mathbf{T}}^k \subset \mathbf{T}$ de tarefas pré requisitos à tarefa t_k, sua posição $t_p^k \in \mathcal{W}$ e seu tipo $t_\tau^k \in \mathcal{T}_{\mathbf{T}}$. O conjunto $\mathcal{T}_{\mathbf{T}}$ de tipos de tarefas também deve ser previamente definido.

Os tipos das tarefas $\mathcal{T}_{\mathbf{T}}$ e dos robôs $\mathcal{T}_R$ são utilizados para determinar a função de esforço $\mathbf{e} : \mathcal{T}_{\mathbf{T}} \times \mathcal{T}_R \rightarrow \mathbb{R}^+$ que determina a quantidade de esforço necessário para que um robô de um tipo $\tau_R \in \mathcal{T}_R$ complete uma tarefa de tipo $\tau_{\mathbf{T}} \in \mathcal{T}_{\mathbf{T}}$. Esse esforço deve representar o tempo aproximado de execução da tarefa, dado que o robô esteja na posição para realizá-la. Caso seja impossível para um robô cumprir uma dada tarefa, sua função de esforço deve retornar ∞. Caso seja desejado, uma função de recompensa $\mathbf{r} : \mathcal{T}_{\mathbf{T}} \rightarrow \mathbb{R}^+$ pode ser derivada dos tipos de tarefas existentes. Essa função deve ser diretamente proporcional a quão vantajoso para o sistema é a realização daquele tipo de tarefa.

Além disso, para melhor definir o nosso ambiente $\mathcal{W}$ será necessário a definição de três coisas. A primeira é a área de busca $\mathcal{W}_{\mathcal{B}} \subset \mathcal{W}$, que é uma subárea do ambiente na qual se deseja limitar a busca. Isso pode facilitar bastante a análise do problema e diminuir o tempo de busca. A segunda é área obstruída $\mathcal{W}_O \subset \mathcal{W}$, ela define todos os obstáculos atualmente conhecidos pelo problema, ela pode ser atualizada durante a execução. A terceira é a função de conectividade $\mathfrak{C} : \mathcal{W} \rightarrow [0, 1]$, essa é uma função que fornece a probabilidade de existir conexão em uma determinada área do ambiente. Modelar a conexão dessa maneira permite que exista uma dissociação entre o algoritmo que utiliza a função de conectividade e sua forma real. Como poucas coisas podem ser assumidas desse modelo, a solução não poderá aproveitar de nenhuma particularidade de um dado modelo e, assim, ser mais generalista.

Após a apresentação dos principais modelos, pode-se pensar em uma maneira de resolver o problema. Para resolvê-lo, o planejamento será dividido em três etapas: o planejamento de missão, a coordenação de missão e a execução do movimento. Essas três etapas devem ser executadas de maneira paralela no sistema e podem ocorrer inclusive em locais diferentes. A configuração escolhida para idealização do projeto de assistentes laboratoriais é a mostrada na Figura 3.1. Algum computador autorizado gera uma nova missão m_{nova}. Essa missão é enviada para um servidor na nuvem que é responsável por realizar o planejamento da missão. A coordenação de missão recebe o planejamento e adiciona passos necessários para garantir a conexão e movimentação colaborativa dos robôs. Esse plano mais detalhado é enviado para cada um dos robôs. Cada um dos robôs tentará executar o plano reagindo a mudanças do ambiente, tanto mudanças físicas como mudanças de conexão. Esses robôs durante a execução podem obter novas informações sobre o ambiente e, dessa forma, realimentar o planejamento de missão. Que pode, por sua vez, gerar um plano melhor e reenviar para a execução de movimento. Toda a parte de localização e posicionamento não será tratada por essa parte. Será assumido neste trabalho que existe um sistema de SLAM (*Simultaneous Location and Mapping)* funcional em cada um dos robôs, ou até mesmo integrado no sistema como um todo.

Agora será analisado com um maior detalhamento cada uma dessas etapas e quais são seus deveres e requisitos. Iniciando pelo planejamento de missão, ele tem duas finalidades principais, gerar os pontos objetivos por onde cada robô deve passar e determinar a ordem de execução das tarefas. Para fazer isso, ele utiliza um modelo do ambiente que vive em constante atualização. Essa etapa é mais deliberativa e pode ser um pouco mais demorada e possuir um maior peso computacional, por isso foi optado por ser executado na nuvem diminuindo a carga de execução em cada um dos robôs.

A coordenação de missão também é um processo mais deliberativo e custoso. Por esse motivo, ela também foi alocada para ser processada na nuvem. A coordenação de missão é responsável por pegar os pontos objetivos gerados pelo planejador e criar trajetórias para cada um dos robôs de uma forma que eles mantenham a conexão entre si e que não colidam com obstáculos conhecidos ou uns com os outros.

Essas duas etapas juntas formam um planejamento mais global. Esse planejamento é feito em uma escala maior, sem se preocupar com detalhes muito pequenos no ambiente, ou com dinamismos no mesmo, como por exemplo, pessoas se movimentando, objetos mudando de lugar, mudanças nos canais de comunicação, movi-

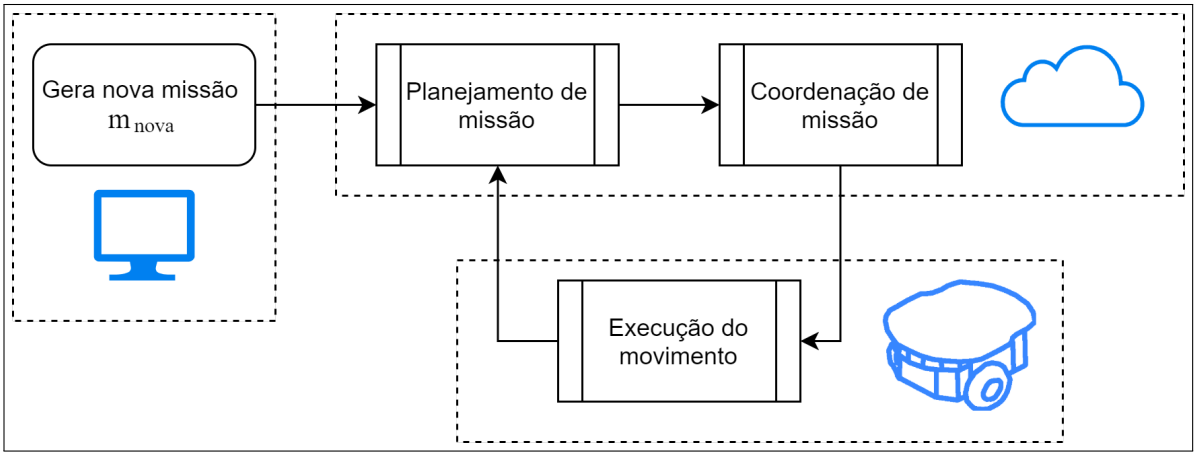


Figura 3.1: Diagrama de execução das diferentes etapas do planejamento. Usuários podem criar novas missões. Um servidor na nuvem prepara e coordena a missão, que é executada pelo robôs.

mentação dos outros robôs, entre outras coisas. O planejamento global é utilizado como um direcionamento das ações, guiando os robôs a realizar as tarefas necessárias. Esse planejamento não precisa ser seguido de maneira rígida, abrindo margem para que mudanças em sua execução sejam feitas, de forma a se adaptar a mudanças no ambiente.

A última etapa desse processo é a execução do movimento. Nela, cada robô é responsável por controlar seu próprio movimento com o objetivo de manter a sua conexão, e de seguir a trajetória planejada, desviando de obstáculos desconhecidos anteriormente e, inclusive, executando alguma rotina para recuperar a conexão em caso de perda. Mesmo que o objetivo deste trabalho seja manter uma conexão contínua, acidentes acontecem e, por isso, é necessário incluir a rotina de recuperação. A etapa de execução do movimento é mais reativa comparada as outras duas. Isso quer dizer que ela tenta utilizar os dados coletados em tempo real, pelos sensores em cada um dos robôs, para reagir e mudar seu comportamento. Isso tudo é feito para manter suas duas principais metas: manter a conexão e evitar colisões.

Essa etapa é conhecida como o planejamento local. Ela se preocupa com dinamismos e novas informações e tenta circundar e evitar cenários indesejáveis, fazendo pequenas alterações locais no planejamento global, sem necessitar de um recálculo completo do plano. Isso economiza processamento e permite que o sistema seja mais robusto e flexível para mudanças no ambiente, e até mesmo para falhas no modelo inicial.

Como dito anteriormente, os robôs executando o movimento coletam e processam novos dados. Esses dados se transformam em mapas que representam melhor o ambiente tanto no aspecto físico espacial, como no aspecto de frequências e canais de comunicação. Isso quer dizer que os robôs em campo podem obter métricas utilizadas para melhorar os modelos utilizados para previsão de conexão do sistema com a base central. Esses novos modelos de mapa físico e de conexão, juntamente com os dados que os geraram, são enviados de volta para o planejamento de missão. Ele, por sua vez, utiliza isso para melhorar seus próprios modelos e, assim, poder gerar planos mais acurados no futuro.

Acredita-se que esse arcabouço apresentado na Figura 3.1 seja capaz de resolver, não apenas o problema dos assistentes laboratoriais, mas também outros cenários mais complexos de CAMP. Ele possui as vantagens de possuir um planejamento global capaz de guiar os robôs para seus alvos sem necessitar de uma exploração extensiva do ambiente, caso se possua algum mapa básico, ou informação a priori do ambiente. Ele também é

capaz de se adaptar ao ambiente e aprender modificações que ocorrem nele. A decisão de deixar as partes mais pesadas do processamento para servidores em nuvem, permite que os robôs tenham mais processamento livre para executar suas ações. Fazendo com que os robôs precisem ter uma conexão com esses servidores na maior parte do tempo. Isso, no entanto, não é um problema aqui, pois uma conexão contínua já era um pré requisito do problema.

Infelizmente, devido a restrição de tempo, não foi possível desenvolver todas as partes necessárias para que as três etapas ficassem completas. Apenas as duas primeiras, planejamento de missão e coordenação de missão, foram completamente finalizadas. Ideias foram geradas para a execução do movimento, mas uma análise mais profunda e detalhada dela não foi possível.

O planejamento de missão foi dividido em duas partes, Seções 3.3 e 3.4. A primeira calcula se é possível realizar uma tarefa com o número de robôs disponíveis, além de calcular a posição onde os robôs auxiliares precisariam ficar para ajudar o robô principal a executar a tarefa. Essas posições auxiliares são chamadas de tarefas-α. Esse nome vem das formas-α que têm um papel fundamental no seu cálculo. A segunda parte utiliza uma heurística para determinar o custo de execução das tarefas, e utiliza isso para decidir qual delas executar.

A coordenação de missão utiliza as tarefas-α para calcular uma maneira de executar a missão escolhida pelo planejador sem perder conexão, Seção 3.5. O coordenador utiliza algoritmos de busca em árvores, como o RRT e o RRT*, para gerar um plano que possa ser executado por cada um dos robôs.

A execução do movimento, a princípio, deve trabalhar para a manutenção de comunicação. Ela faria isso de duas maneiras, uma preventiva e uma reativa. A maneira preventiva seria por meio do mapeamento dos canais de comunicação fixos. Como os roteadores são fixos e o ambiente apesar de dinâmico, não muda o suficiente para justificar grandes mudanças no perfil de radiação dos mesmos, é possível propor algoritmos que mapeiem os canais e gerem um mapa de conexão. Esses algoritmos teriam como base os trabalho de [8], que deriva os parâmetros dos modelos probabilísticos do canal de comunicação a partir de uma pequena quantidade de amostras, e o trabalho em [41] que tenta construir um mapa baseado em Campos Gaussianos aleatórios, como os apresentados na Figura 3.2, a partir de um modelo probabilístico do canal ajustado com as amostras obtidas durante a execução.

Os robôs estão em constante movimento, portanto a estimação de um mapa que represente seu perfil de radiação seria muito difícil de obter. Por causa disso, essa segunda etapa é reativa, ela seria baseada no trabalho

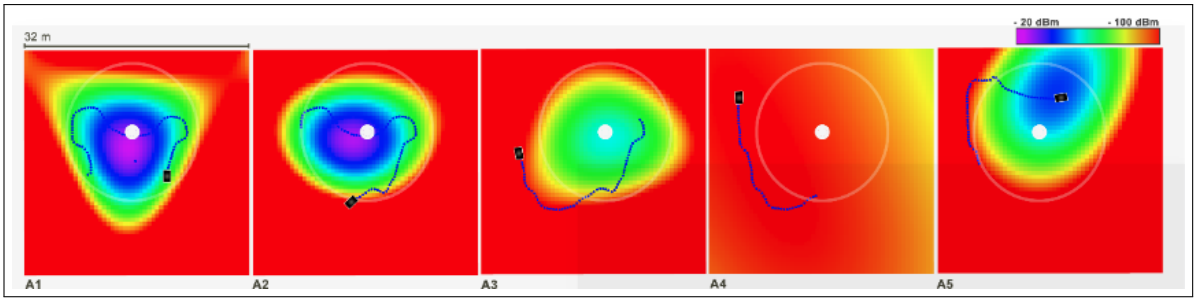


Figura 3.2: Mapa construído a partir de modelo probabilístico do canal, baseado em Campos Gaussianos aleatórios. As cores de fundo representam a intensidade do sinal em uma determinada região, de acordo com a legenda na parte superior direita da imagem. O trajeto pontilhado representa o trajeto que o robô fez durante o mapeamento, e o ponto branco representa a posição real do roteador. Fonte [41].

em [62]. Para isso, a ideia seria utilizar métricas adquiridas de protocolos estabelecidos na área de redes de comunicação, como o OLSR[1], para determinar a qualidade do link. Essa métrica seria utilizada para estimar uma função de conectividade inter-robôs g_{ij}, entre os robôs i e j. Com o auxílio dessa função, a ideia seria utilizar um controlador da seguinte forma

$$\dot{q} = u_i = \underbrace{-k\nabla_i \Phi_i(q_i)}_{\text{Navegar para o alvo}} \underbrace{- \sum_{j \in \Gamma_i} \nabla_i g_{ij}(q_i, q_j)}_{\text{Manter a qualidade do link}}. \tag{3.1}$$

Nessa equação, q_i é a pose do i-ésimo robô; k é uma constate positiva; Φ_i é a função potencial do i-ésimo robô, responsável por guiá-lo a seguir o plano recebido; Γ_i é o conjunto dos robôs próximos ao i-ésimo robô. E como já especificado, q_{ij} são as funções potenciais utilizadas para modelar o comportamento das conexões inter-robôs. Com a sua rota planejada, o executor de movimento faria com que o robô alcançasse cada um dos pontos da rota dentro do tempo estipulado. Como a qualidade do canal de comunicação varia muito, é necessário que o executor reaja, não somente a obstáculos desconhecidos, como às variações inadequadas da rede, e é justamente isso que esse controlador tenta evitar.

É importante ressaltar, novamente, que toda essa parte do executor não chegou a ser devidamente desenvolvida e implementada, ainda, devido a falta de tempo. No entanto, agora e até o final do capítulo, serão discutidos somente o que foi de fato desenvolvido, implementado e testado. Iniciando pela criação de uma linguagem descritiva para a representação desse problema.

3.2.1 Linguagem descritiva de problemas de CAMP

O modelo matemático que descreve a situação deve ser capaz de ser repassado entre computadores. Além disso, várias dessas informações podem ser reutilizadas diversas vezes em cenários ligeiramente diferentes. Dessa forma, foi desenvolvida uma linguagem descritiva para representar essa classe de problemas. Essa linguagem foi baseada no JSON[2]. O JSON é um padrão de sintaxe textual que facilita a troca de dados entre linguagens de programação. Ele foi escolhido por possuir suporte nativo, ou por meio de bibliotecas, em quase todas as linguagens de programação mais importantes. Além de possuir suporte nativo em muitos bancos de dados. Outra grande vantagem do JSON é que ele é facilmente tanto por máquinas como por humanos, facilitando a sua escrita e compreensão.

O problema é representado por meio de um arquivo JSON, como o da Figura 3.3. Este arquivo possui campos que representam os modelos matemáticos necessários para descrever o problema que se deseja abordar. O JSON funciona no esquema "chave : valor". Cada entrada no arquivo é separada por vírgulas, a chave é sempre uma *string* de texto entre aspas. O valor da chave fica logo após a declaração da chave, depois dos ":". O valor pode ser uma *string,* um booleano, um valor numérico, um valor considerado como nulo, um outro objeto JSON ou um vetor de alguns desses outros grupos.

O JSON que descreve o problema é composto por dez campos. Cada um deles corresponde a uma peça chave do modelo matemático apresentado. O campo *"robot_types"* representa o conjunto de tipos de robôs $\mathcal{T}_R$. Assim como ele, o campo "*task_types*" guarda o conjunto de tipos de tarefas $\mathcal{T}_\mathbf{T}$. O campo "*effort_function*" é a função de esforço $\mathbf{e}$. O próximo campo "*robots*" contém o conjunto de robôs $\mathbf{R}$ que irão participar do

[1] https://www.ietf.org/rfc/rfc3626.txt
[2] JSON Standard 2017

```
{
        "robot_types":[],
        "task_types":[],
        "effort_function":{},
        "reward_function":{},
        "robots":[],
        "tasks":[],
        "missions":[],
        "search_area":{},
        "obstructed_area":{},
        "connectivity_function:":{}
}
```

Figura 3.3: Estrutura de um JSON que representa um problema de CAMP.

planejamento. O campo "*tasks*" contém todas as tarefas do conjunto $\mathbf{T}$. Em "*missions*" são representadas todas as missões $\mathbf{M}$ do problema. Para representação do ambiente existem os campos "*search_area*", "*obstructed_area*" e "*connectivity_function*". Eles representam, respectivamente os modelos da área de busca $\mathcal{W}_B$, da área obstruída $\mathcal{W}_O$ e da função de conectividade $\mathfrak{C}$.

Foi implementado um *parser* que verifica se o modelo contido no arquivo JSON faz sentido. Ele é capaz de analisar o modelo e identificar inconsistências. Agora será analisado com detalhes cada um dos dez campos necessários, explicando como eles podem ser representados nessa linguagem e que tipo de verificação deve ser feita.

Tipos de robôs ou "*robot_types*" - $\mathcal{T}_R$

Os tipos de robôs $\mathcal{T}_R$, ou "*robot_types*", devem ser representados como um vetor de *strings*, não deve ocorrer repetições. Na sintaxe JSON, um vetor é caracterizado por itens do mesmo tipo agrupados entre "[]" e separados por vírgulas. Como mostrado na Figura 3.4. A única verificação necessária é para garantir que não existem tipos duplicados.

```
{
        "robot_types": ["type 1", "type 2", ..., "type n"]
}
```

Figura 3.4: JSON dos tipos de robôs.

Tipos de tarefas ou "*task_types*" - $\mathcal{T}_\mathbf{T}$

Os tipos de tarefas $\mathcal{T}_\mathbf{T}$, ou "*task_types*", seguem a mesma convenção dos tipos robôs. O conjunto é representado por um vetor de *strings* sem repetições. Um exemplo pode ser visualizado na Figura 3.5. Novamente, a única verificação necessária é identificar se alguma *string* aparece mais de uma vez no vetor.

```
{
        "task_types": ["type 1", "type 2", ..., "type n"]
}
```

Figura 3.5: JSON dos tipos de tarefas.

Função de esforço ou "*effort_function*" - $\mathbf{e} : \mathcal{T}_\mathbf{T} \times \mathcal{T}_\mathcal{R} \to \mathbb{R}^+$

A função de esforço $\mathbf{e} : \mathcal{T}_\mathbf{T} \times \mathcal{T}_\mathcal{R} \to \mathbb{R}^+$, ou "*effort_function*", é um objeto JSON que mapeia os tipos de tarefas e tipos de robôs para o seu valor de esforço. Levando em conta que a operação $|A|$ denota o número de elementos do conjunto A, esse mapeamento deve conter exatamente $|\mathcal{T}_\mathbf{T}| \times |\mathcal{T}_R|$ elementos, como na Figura 3.6. Caso uma tarefa seja impossível de ser realizada por um robô, é atribuída a ela um valor infinito. Para isso se utiliza a *string* "*infinity*". O *parser* verifica se todos os elementos de $\mathcal{T}_\mathbf{T}$ e de $\mathcal{T}_R$ foram utilizados. Analisa se algum tipo indefinido foi utilizado, ou seja, se algum tipo do mapeamento não pertence a $\mathcal{T}_\mathbf{T}$ ou $\mathcal{T}_R$. Além disso, ele também averiguá se todos os valores são numéricos ou infinito.

```
{
        "effort_function":{
                "T1":{"R1": 0.60, "R2": 0.10,       "R3": 0.34},
                "T2":{"R1": 0.43, "R2": 0.78,       "R3": 0.47},
                "T3":{"R1": 1.00, "R2": "infinity", "R3": 0.30}
        }
}
```

Figura 3.6: JSON dos tipos de tarefas.

Função de recompensa ou "*reward_function*" - $\mathbf{r} : \mathcal{T}_\mathbf{T} \to \mathbb{R}^+$

A função de recompensa $\mathbf{r} : \mathcal{T}_\mathbf{T} \to \mathbb{R}^+$, ou "*reward_function*", é um JSON que mapeia os tipos de tarefas ao seu valor de recompensa. Esse mapeamento deve conter exatamente $|\mathcal{T}_\mathbf{T}|$ elementos, como na Figura 3.7. O *parser* deve fazer algumas verificações para esse objeto também. Primeiro ele deve garantir que todos os tipos em $\mathcal{T}_\mathbf{T}$ estão representados na função e que nenhum tipo indefinido tenha sido usado. Por fim, ele deve garantir todos os valores sejam numéricos.

```
{
        "reward_function":{
                "T1":0.7, "T2":3.9, "T3":4.1
        }
}
```

Figura 3.7: JSON da função de recompensa.

Robôs ou "*robots*" - $\mathbf{R}$

Os robôs $\mathbf{R}$, ou "*robots*", são descritos por um vetor de objetos JSON. Cada um dos objetos desse vetor descreve, cada qual, um robô. Esses objetos de robôs seguem o padrão da Figura 3.8. Esse objeto representa um robô $r_i \in \mathbf{R}$ e possui três itens em sua estrutura, um item a mais do que as duas características necessárias r_τ^i e r_p^i. Esse item extra é o de chave "*name*" e é um espaço para nomear o robô. Esse é um passo extra usado apenas para facilitar a identificação e visualização durante simulações. As chaves "*type*" e "*position*" representam, respectivamente, o tipo r_τ^i e a posição r_p^i do robô. O tipo r_τ^i deve ser uma *string* que coincida com algum dos tipos declarados em "*robot_types*", dessa forma garantindo que $r_\tau^i \in \mathcal{T}_\mathcal{R}$. Essa é a primeira verificação que o *parser* realiza. A posição r_p^i é um categoria a parte de objetos JSON. Ela depende do tipo especificado em seu campo "*type*", sendo no exemplo dado bidimensional. O *parser* também deve verificar se o tipo de posição utilizado em toda a representação do problema é consistente.

```
{
        "name": "Nome do robo",
        "type": "Tipo",
        "position" : {
                "type": "2d",
                "x": 0.0,
                "y": 0.0
        }
}
```

Figura 3.8: JSON de um robô.

Tarefas ou "*tasks*" - $\mathbf{T}$

As tarefas $\mathbf{T}$ ou "*tasks*" são descritas por um vetor de objetos JSON. Cada um dos objetos desse vetor descreve, cada qual, uma tarefa. Esses objetos de tarefas seguem o padrão da Figura 3.9. Esse objeto representa uma tarefa $t_k \in \mathbf{T}$ e possui quatro itens em sua estrutura, um item a mais do que as três características necessárias $t_\mathbf{T}^k$, t_p^k e t_τ^k. Esse item extra é o de chave "*id*" e é um espaço utilizado para identificar uma tarefa. Esse é um passo extra usado apenas para facilitar a identificação e visualização durante simulações. As chaves "*type*", "*position*" e "*prerequisites*" representam, respectivamente, o tipo t_τ^k, a posição t_p^k e o conjunto de tarefas pré requisitos $t_\mathbf{T}^k$ da tarefa. O tipo t_τ^k deve ser uma *string* que coincida com algum dos tipos declarados em "*task_types*", dessa forma garantindo que $t_\tau^k \in \mathcal{T}_\mathbf{T}$. Essa é a primeira verificação que o *parser* realiza. A posição t_p^k deve obedecer as mesmas regras que a posição do robôs segue. O *parser* deve analisar se os seus tipos são compatíveis. O conjunto de de pré requisitos $t_\mathbf{T}^k$ é apresentado como um vetor de *strings*. Ele deve conter o id de todas as tarefas que precisam estar completas antes dela, sem considerar os pré requisitos indiretos. Isso quer dizer que se t_a é pré requisito direto de t_b, que por sua vez é pré requisito direto de t_c, apenas t_b deve aparecer no vetor de pré requisitos de t_c. Por mais que t_a seja um pré requisito indireto, ele não aparece na lista de t_c. O *parser* verifica se existem referências circulares, ou seja, se t_a é de algum jeito pré requisito indireto de t_a. Caso elas existam, uma mensagem de erro é gerada. Esse tipo dependência não é permitida nesse problema. Além disso, também é verificado se alguma tarefa indefinida foi utilizada como pré requisito.

```
{
        "id": "id",
        "type": "Tipo",
        "position": {
                "type":"2d",
                "x": 0.0,
                "y": 0.0
        }
        "prerequisites": ["id1", "id2", ..., "idN"]
}
```

Figura 3.9: JSON de um tarefa.

Missões ou "*missions*" - M

As missões $\mathbf{M}$, ou "*missions*", são descritas por um vetor de objetos JSON. Cada um dos objetos desse vetor descreve, cada qual, uma missão. Esses objetos de missões seguem o padrão da Figura 3.10. Esse objeto representa uma missão $m_j \in \mathbf{M}$ e possui quatro itens em sua estrutura, um item a mais do que as três características necessárias $m^j_{\mathbf{T}}$, m^j_π e $m^j_{\mathbf{0}}$. Esse item extra é o de chave "*id*" e é um espaço para identificar a missão. Esse é um passo extra usado apenas para facilitar a identificação e visualização durante simulações. As chaves "*priority*", "*deadline*" e "*mission_tasks*" representam, respectivamente, a prioridade m^j_π, o prazo $m^j_{\mathbf{0}}$ e o conjunto de tarefas $m^j_{\mathbf{T}}$ que fazem parte daquela missão. O *parser* verifica se a prioridade e o prazo são números, se a missão possui tarefas associadas a elas e se as tarefas associadas a ela pertencem a $\mathbf{T}$. Para isso, o conjunto $m^j_{\mathbf{T}}$ é representado como um vetor de *strings*, as quais devem ser *ids* de tarefas declaradas em "*tasks*". Outra coisa que é averiguada é, se existem tarefas em $\mathbf{T}$ que não pertencem a nenhum $m^j_{\mathbf{T}}$ e se existe alguma intersecção entre os diferentes $m^j_{\mathbf{T}}$, em outras palavras,

$$\bigcap_{j=1}^{n_m} m^j_T = \emptyset \text{ e} \tag{3.2}$$

$$\bigcup_{j=1}^{n_m} m^j_T = \mathbf{T}. \tag{3.3}$$

Se alguma dessas duas equações não for verdade, o *parser* acusa um erro de modelo.

```
{
        "id": "id",
        "priority": numero,
        "deadline": numero,
        "mission_tasks": ["task_id_1", "task_id_2", ..., "task_id_n"]
}
```

Figura 3.10: JSON de um missão.

Área de busca ou "*search_area*" - $\mathcal{W}_B$

A área de busca $\mathcal{W}_B$, ou "*search_area*", deve marcar o limite de onde deve ocorrer o planejamento. No futuro se deseja que esse limite possa ser dado por meio de arquivos externos, como imagens ou descritores mais exatos e extensos, porém como ponto inicial foi criado uma classe de objetos JSON que representam formas geométricas simples, como a presente no JSON da Figura 3.11. Nesse exemplo a forma geométrica é um retângulo, esse retângulo é definido por uma coordenada de posição, representada por um JSON de posicionamento, juntamente com uma largura e uma altura. Nesse caso, o *parser* analisa o tipo da busca de área, caso seja um forma geométrica válida (por enquanto apenas círculos e retângulos) ele verifica se a posição faz sentido para o sistema de coordenadas utilizado nos outros lugares e averígua se a largura e a altura são não negativas.

```
{
        "search_area": {
                "type":"rectangle",
                "bottom_left": {
                        "type":"2d",
                        "x": 0.0,
                        "y": 0.0
                },
                "width": number,
                "height": number,
        }
}
```

Figura 3.11: JSON de uma área de busca.

Área obstruída ou "*obstructed_area*" - $\mathcal{W}_O$

A área obstruída $\mathcal{W}_O$, ou "*obstructed_area*", deve representar todos os obstáculos atualmente conhecidos. A ideia é que ela possa ser representada, também, por meio de arquivos externos, como mapas bidimensionais, ou modelos de maior dimensionalidade, no entanto, inicialmente são utilizadas um conjunto dos objetos JSON de formas geométricas descritos anteriormente. Um exemplo de uma área obstruída pode ser visualizada na Figura 3.12. Nesse exemplo existem um retângulo e um círculo de obstáculo. Para dizer que formas geométricas serão utilizadas, a chave "*type*" deve conter o valor "*geometric*". Quando isso acontece, deve existir também um campo "*areas*". Esse campo tem como valor um vetor de objetos JSON do tipo geométrico. O *parser* faz as mesmas análises realizadas para a área de busca, a única diferença é que ele deve verificar cada um dos elementos desse vetor.

É possível perceber que em ambientes complexos, que exijam muitas formas geométricas para descrever, esse item do objeto JSON irá ficar muito grande. Por isso, já está nos planos mudar a declaração dessa estrutura para outro arquivo. Assim, no arquivo principal seria modificado o tipo de $\mathcal{W}_O$ e em um outro campo ficaria o nome do arquivo onde está a declaração do espaço obstruído. Isso facilitaria, não apenas a visualização e criação desse arquivo, como a reutilização do mesmo mapa em mais de um problema.

```
{
        "obstructed_area":{
                "type":"geometric",
                "areas":[
                        {
                                "type":"rectangle",
                                "bottom_left": {
                                        "type":"2d",
                                        "x":0,
                                        "y":-1
                                },
                                "width":1,
                                "height":4
                        },
                        {
                                "type":"circle",
                                "center": {
                                        "type":"2d",
                                        "x":2.5,
                                        "y":1.5
                                },
                                "radius":0.5
                        }
                ]
        }
}
```

Figura 3.12: JSON da área obstruída.

Função de conectividade ou "*connectivity_function*" - $\mathfrak{C} : \mathcal{W} \rightarrow [0, 1]$

A função de conectividade $\mathfrak{C} : \mathcal{W} \rightarrow [0, 1]$, ou "*connectivity_function*", pode ser descrita da mesma forma que a área obstruída. Essa é uma simplificação em que as formas geométricas representariam as áreas do espaço nas quais a função de conectividade seria maior que um limiar $\gamma_{\mathfrak{C}}$ mínimo de probabilidade de conexão. No futuro deseja-se encontrar formas de incluir modelos mais precisos de modelagem de canais de comunicação, assim como mapas de conexão, em que a intensidade de pixeis seria correspondente a probabilidade de haver conexão naquele local. O JSON para a função de conectividade séria quase idêntico ao da Figura 3.12, mudando a chave inicial para "*connectivity_function*".

O Apêndice A possui um exemplo de um arquivo de descrição completo. O arquivo descreve uma parte do primeiro andar do SG-11, prédio onde fica localizado o LARA. É incluído, também, uma imagem (Figura A.1) da representação visual do problema. Foram incluídos três robôs e cinco tarefas. As tarefas são divididas entre duas missões. É considerado apenas uma região circular de conexão. A seguir, na próxima seção, serão detalhados os passos do cálculo dos pontos da missão, tendo como objetivo principal manter uma conexão

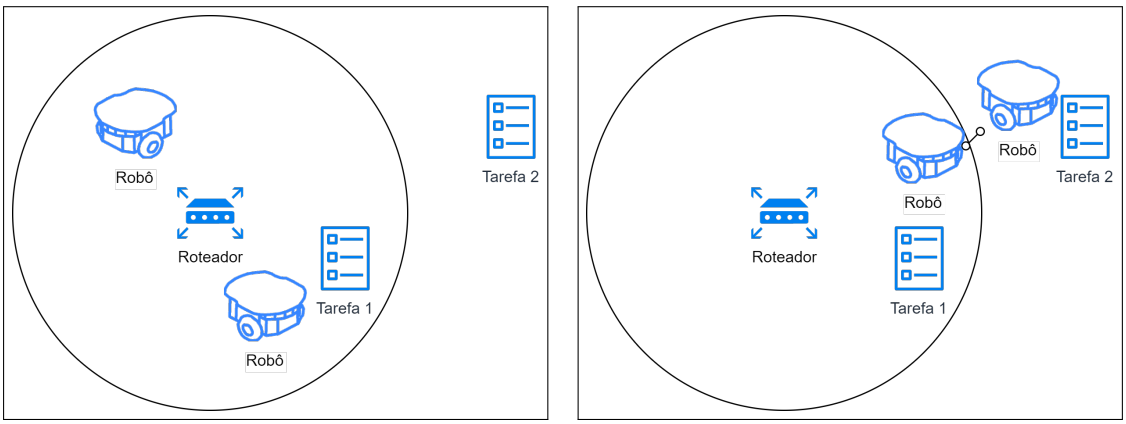
contínua com a base, utilizando, se possível, o menor número de robôs.

3.3 CALCULANDO AS TAREFAS AUXILIARES: TAREFAS-α

O planejamento de movimento convencional, geralmente, se preocupa com duas coisas: deslocar o robô de um ponto A até um ponto B; e evitar que ele sofra colisões durante esse trajeto. Nesse cenário, determinar se um ponto é alcançável faz parte do planejamento, e não faz sentido separar a etapa de planejamento do movimento, da etapa de determinar a alcançabilidade desse ponto B, já que os algoritmos para ambos seriam bem parecidos. Porém, isso muda quando a conectividade de um robô deve ser mantida durante o seu movimento. Nesse caso, determinar um plano de movimentação completo se torna muito mais custoso, e é possível propor um algoritmo que analise a alcançabilidade de uma tarefa.

Para entender melhor o problema será analisado, inicialmente, um caso simples, como o da Figura 3.13. Nela existem dois robôs e duas tarefas, uma dentro e uma fora da zona de conexão, representada pelo círculo centrado no roteador. A tarefa 1 se encontra dentro da região de conexão, portanto sua alcançabilidade pode ser determinada como no caso normal onde não é necessário se preocupar com conexão. Em outras palavras, caso não existam obstáculos físicos, um robô pode simplesmente se deslocar pelo interior da região de conexão e realizar a tarefa, como mostrado na Figura 3.13a. Já a tarefa 2 se encontra fora da zona de conexão, e por causa disso, um robô não pode simplesmente se deslocar até ela, caso contrário ele perderia conexão com o roteador. Para evitar isso, um robô pode servir como roteador para o outro, fornecendo conexão para que o primeiro realize a tarefa, como exemplificado na Figura 3.13b. Nela, o robô dentro da zona de conexão serve como uma ponte para o roteador. Esse ponto onde o robô auxiliar deve ficar é o que será chamado de tarefa-α. O algoritmo proposto nesta seção mostra uma maneira de calcular essas tarefas-α e explica como elas são utilizadas para determinar a alcançabilidade de uma tarefa, considerando o número de robôs disponíveis.

O cálculo das tarefas-α utiliza os princípios do planejamento baseado em amostras, Seção 2.4.4. Os modelos da área obstruída $\mathcal{W}_O$ e a função de conectividade $\mathfrak{C} : \mathcal{W} \rightarrow [0, 1]$ são utilizados para evitar a construção



(a) Realizando tarefa dentro da zona de conexão

(b) Realizando tarefa fora da zona de conexão

Figura 3.13: Um cenário onde uma tarefa se encontra dentro, enquanto a outra se encontra fora da zona de conexão. Como determinar se é possível realizar a segunda tarefa?

explícita da região $\mathcal{C}_{obs}$. Como a abordagem escolhida foi a de processamento em nuvem, e como o mesmo ambiente é utilizado para o cálculo de tarefas-α para diversas tarefas, uma abordagem envolvendo *roadmaps* foi escolhida. Como explicado anteriormente, esses algoritmos são divididos em duas partes: a fase de preprocessamento e a fase de busca. Na fase de preprocessamento um *roadmap* do ambiente é construído, para que posteriormente todas as buscas sejam executadas sobre esse *roadmap* já estabelecido, na fase de busca.

Esse *roadmap* pode ser representado como um grafo não direcionado de busca $\mathcal{G}(V, E)$, em que V é o conjunto de vértices que representam as configurações em $\mathcal{C}_{free}$, e que E são as aresta que representam caminhos contínuos válidos entre dois vértice de V. Essa aresta deve estar completamente contida em $\mathcal{C}_{free}$ e não deve sobrepor nenhuma outra aresta, a não ser nas suas extremidades. Para agilizar futuras consultas a esse *roadmap*, o algoritmo de Dijkstra (Seção 2.4.6) é utilizado para calcular a função de custo $\mathcal{D} : V \to \mathbb{R}$. Essa função representa o custo para se alcançar o vértice $v \in V$ mais próxima que obedeça a seguinte regra, $\mathfrak{C}(v) > \gamma_{\mathfrak{C}}$, ou seja, a probabilidade de existir conexão naquele vértice é maior do que o limiar $\gamma_{\mathfrak{C}}$ mínimo de conexão. Em outras palavras, o vértice $v \in V$ deve estar dentro de uma zona de conectividade.

Ao se unir o grafo de busca de conectividade $\mathcal{G}_{con}(V, E)$ com a função de custo $\mathcal{D} : V \to \mathbb{R}$ associada a esse grafo, se obtém uma estrutura, aqui conhecida como mapa de busca $\mathcal{M}(V, E, \mathcal{D})$. Esse mapa $\mathcal{M}(V, E, \mathcal{D})$ será utilizado, posteriormente, como o *roadmap* para consultas futuras. O Algoritmo 3.1 descreve uma maneira de construir o mapa de busca $\mathcal{M}$.

O Algoritmo 3.1 corresponde a etapa de preprocessamento do planejamento baseado em amostra com *roadmaps*. O primeiro passo para construção de $\mathcal{M}$ é a determinação de todas as configurações de borda que definem a zona de conectividade $\mathcal{P}_{con}$. O formato da função de conectividade $\mathfrak{C} : \mathcal{W} \to [0, 1]$ não é conhecido a priori, ela pode ser qualquer coisa, desde um conjunto de formas geométricas a equações probabilísticas complexas, por esse motivo, a função DETERMINAR_BORDA_DE_CONECTIVIDADE (Algoritmo 3.2) deve ser capaz de construir uma representação válida da informação contida em $\mathfrak{C}$. A abordagem proposta para construção dessa representação é a amostragem aleatória. Por meio dessa amostragem, é possível obter um formato aproximado da zona de conectividade original. Para isso, deve-se escolher $n_{con} \in \mathbb{N}$ amostras aleatoriamente do ambiente. Essas amostras devem respeitar as seguintes condições:

- o valor de sua função de conectividade $\mathfrak{C}$ deve ser maior que o mínimo $\gamma_{\mathfrak{C}}$;
- elas devem estar pelo menos a uma distância mínima r_{dm} de qualquer outra amostra.

Caso uma amostra não atenda ambos os critérios, ela é reamostrada um máximo de n_t vezes. Se nenhuma dessas tentativas funcionar, a amostragem fica com menos do que n_{con} amostras. A distância mínima entre

Algoritmo 3.1 Construindo um mapa de busca $\mathcal{M}(V, E, \mathcal{D})$.

1: **função** CONSTUIR_MAPA_DE_BUSCA($\mathfrak{C}$, $\mathcal{W}_O$)
2: $\mathcal{P}_{con} \leftarrow$ DETERMINAR_BORDA_DE_CONECTIVIDADE ($\mathfrak{C}$)
3: $\mathcal{P}_{free} \leftarrow$ GERAR_AMOSTRAS_VALIDAS ($\mathfrak{C}, \mathcal{W}_O$)
4: $\mathcal{P} \leftarrow$ UNIR ($\mathcal{P}_{con}, \mathcal{P}_{free}$)
5: $\mathcal{G}_{con} \leftarrow$ CONECTAR_AMOSTRAS ($\mathcal{P}, \mathfrak{C}, \mathcal{W}_O$)
6: $\mathcal{D} \leftarrow$ DIJKSTRA ($\mathcal{G}_{con}, \mathcal{P}_{con}$)
7: $\mathcal{M} \leftarrow$ CRIAR_MAPA_DE_BUSCA ($\mathcal{G}_{con}, \mathcal{D}$)
8: **retorna** $\mathcal{M}$
9: **fim função**

Algoritmo 3.2 Determinando a região da borda de conectividade.

função DETERMINAR_BORDA_DE_CONECTIVIDADE($\mathfrak{C}$)
    $n_{con} \leftarrow 1000$
    $n_t \leftarrow 100$
    $\mathcal{P} \leftarrow \emptyset$
    $r_{dm} \leftarrow$ RAIO_DISTÂNCIA_MÍNIMA(n_{con})
    para $i \leftarrow 0$ **até** n_{con} **faça**
        para $j \leftarrow 0$ **até** n_t **faça**
            $q_{\text{novo}} \leftarrow \alpha(i)$
            se ($\mathfrak{C}(q_{\text{novo}}) > \gamma_{\mathfrak{C}}$) **e** ($\nexists p \in \mathcal{P} | \rho(p, q_{\text{novo}}) < r_{dm}$) **então**
                $\mathcal{P} \leftarrow \mathcal{P} \cup \{q_{novo}\}$
                pula para próxima amostra
            fim se
        fim para
    fim para
    $\mathcal{P}_{con} \leftarrow$ FORMA_$\alpha(\mathcal{P}, 10r_{dm})$
    retorna $\mathcal{P}_{con}$
fim função

amostras r_{dm} é dada pela função RAIO_DISTÂNCIA_MÍNIMA. No caso de amostragens em duas dimensões, pode-se considerar que a densidade máxima de preenchimento de uma área planar por meio de círculos é dada por η_h, quando os círculos se posicionam de maneira hexagonal [63]. Em que,

$$\eta_h = \frac{\pi}{2\sqrt{3}}. \tag{3.4}$$

Considerando $A(\mathcal{W}_B)$ como a área da região de busca, então a área que de fato será ocupada por círculos de raio r_{dm} é $A(\mathcal{W}_B)\eta_h$. Dividindo isso pela área do círculo πr_{dm}^2 se obtém o número aproximado de círculos

$$n_{con} = \frac{A(\mathcal{W}_B)\eta_h}{\pi r_{dm}^2} \tag{3.5}$$

que cabem nessa região. Trabalhando essa equação 3.5, se obtém um fórmula para determinar o raio de distância

$$r_{dm} = \sqrt{\frac{A(\mathcal{W}_B)}{2\sqrt{3}n_{con}}} \tag{3.6}$$

para se ter aproximadamente n_{con} amostras na área de busca.

Agora que as amostras $\mathcal{P}$ foram obtidas, formas-α (Seção 2.4.7) são construídas a partir desse conjunto. Como explicado anteriormente, essa classe de formas foi escolhida pelo seu benefício dual de possuírem um fundação matemática sólida e por serem relativamente simples de calcular. A função FORMA-α retorna todas as amostras α-extremas desse conjunto de amostras. Como explicado na Seção 2.4.7, amostras α-extremas são as amostras que geram a forma-α. Foi utilizado um valor de $\alpha < 0$, pois, assim, é possível capturar formas mais diversas do que com α positivo. Um valor que obteve bons resultados experimentais foi $-\alpha = 10r_{dm}$. As amostras que são α-extremos são armazenadas no conjunto $\mathcal{P}_{con}$. Essas amostras serão os alvos de busca do algoritmo, e é por esse motivo que as tarefas auxiliares geradas a partir dessa busca são chamadas de tarefas-α.

O próximo passo é gerar um conjunto de amostras válidas em $\mathcal{C}_{free}$. Essas amostras devem conseguir representar o ambiente com uma resolução adequada. A função GERAR_AMOSTRAS_VALIDAS faz justamente isso, com o auxílio da função de conectividade $\mathfrak{C}$ e da área obstruída $\mathcal{W}_O$. Ela gera $n_{free} \in \mathbb{N}$ amostras utilizando uma estratégia muito similar a do Algoritmo 3.2. As diferenças são que, primeiramente, não é calculado a forma-α dessas amostras, e que uma amostra q só é escolhida, se ela respeitar três diferentes condições, considerando que $\mathcal{O}(q) : \mathcal{W} \rightarrow \{0, 1\}$ é uma função que diz se uma amostra pertence ou não a região $\mathcal{W}_O$:

- o valor de $\mathcal{O}(q)$ deve ser 0;
- o valor de $\mathfrak{C}(q)$ deve estar abaixo do limiar $\gamma_{\mathfrak{C}}$;
- elas devem estar pelo menos a uma distância mínima r_{dm} de qualquer outra amostra, ou seja, $\nexists p \in \mathcal{P}_{free} | \rho(p, q) < r_{dm}$.

Em que $\mathcal{P}_{free}$ é o conjunto de amostras aleatórias que obedecem essas três condições. Essas condições caracterizam amostras que estejam fora região de conectividade, e que não colidam com nenhum obstáculo. Essas amostras, $\mathcal{P}_{free}$, juntamente com $\mathcal{P}_{con}$ formam o conjunto $\mathcal{P}$, que serve como a base sobre a qual a busca pelas tarefas-α ocorrem.

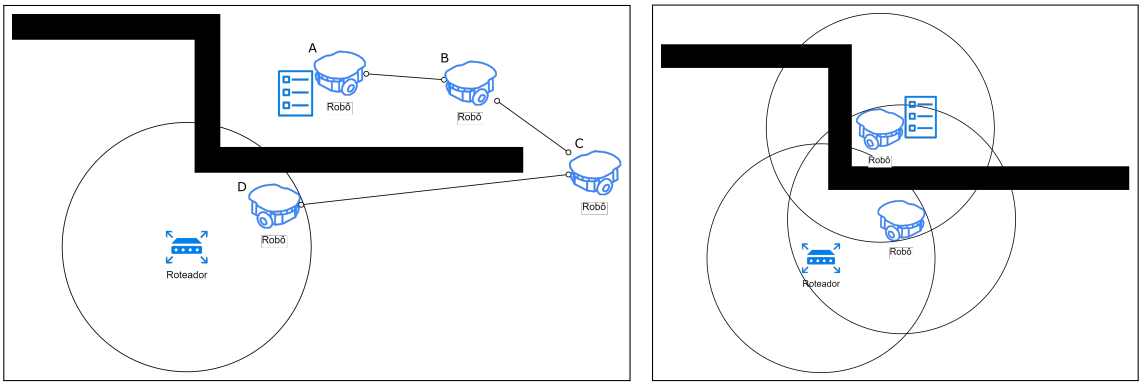
As amostras de $\mathcal{P}$ estão completamente desconectadas uma das outras. Para remediar essa situação a função CONECTAR_AMOSTRAS deve conectar todos as amostras de uma maneira concisa e completa, de forma que todos os possíveis caminhos entre essas amostras estejam representados. Para fazer isso foi utilizada a triangulação de Delaunay, Seção 2.4.7.

Como dito anteriormente, uma triangulação provê um grafo planar de linhas retas $\mathcal{G}(V, E)$, em que nenhum par de arestas de E se intersecta propriamente. Essa triangulação em específico foi escolhida, pois pode ser calculada em um tempo $O(n \log n)$, em que n é o número de vértices em V. E também porque possui duas características interessantes: é a triangulação que maximiza o menor ângulo dos triângulos; e o círculo circunscrito em qualquer triângulo dessa triangulação não possui nenhum ponto de V em seu interior.

Com a triangulação feita, qualquer aresta $e \in E$ que intersecte com $\mathcal{W}_O$, ou com uma região na qual $\mathfrak{C}$ tenha uma valor acima de $\gamma_{\mathfrak{C}}$, é descartada. Assim, se obtém $\mathcal{G}_{con}(V, E)$, um grafo em que $V = \mathcal{P}$, e que E contém todas as arestas que conectam essas amostras de acordo com a triangulação de Delaunay e que não violem as condições apresentadas.

Para calcular a distância de cada vértice $v \in V$ do grafo para a amostra $p_{con} \in \mathcal{P}_{con}$ da zona de conectividade mais próxima, o Algoritmo de Dijkstra (Seção 2.4.6) foi utilizado. As amostras do grafo que pertencem a $\mathcal{P}_{con}$ foram utilizadas como o conjunto $\mathcal{S}$ de entrada para o Algoritmo 2.6. Assim, todos os pontos de V que pertençam a $\mathcal{P}_{con}$ são iniciados com um custo inicial de 0 e todos os outros pontos com um custo inicial de ∞. Dessa maneira, $\mathcal{D}$, que é o mapa de custo resultante do algoritmo de Dijkstra, contém o custo de cada amostra para a amostra mais próxima que faça parte da zona de conectividade. Sendo assim, $\mathcal{G}_{con}(V, E)$ e $\mathcal{D}$ formam, conjuntamente, o mapa de busca $\mathcal{M}(V, E, D)$.

Antes de continuar para a segunda parte do algoritmo de busca é importante definir algumas coisas. Como dito anteriormente, a modelagem de canal de comunicação entre dois robôs em movimento em um ambiente dinâmico pode ser algo muito difícil e custoso. Uma alternativa para lidar com isso é assumir um determinado modelo para o planejamento e ir adaptando durante a execução. Dois dos modelos mais simples para essa modelagem são o radial (Figura 3.14b) e o de visibilidade (Figura 3.14a). O modelo radial assume um disco de tamanho constante. No interior desse disco existe conexão e fora não. O modelo de visibilidade assume que enquanto existe uma linha de visada entre os robôs, existe conexão.



(a) Conexão por visibilidade.

(b) Conexão radial.

Figura 3.14: Arranjos de conexão para realização de uma tarefa.

Pode-se utilizar os exemplos da Figura 3.14 para entender as vantagens e desvantagens de cada um dos modelos. O modelo radial (Figura 3.14b) consegue alcançar e realizar a tarefa com menos robôs do que a sua contrapartida por visibilidade (Figura 3.14a). Esse último precisa de quatro robôs, enquanto o primeiro utiliza apenas dois. Isso acontece, pois o modelo radial admite uma permeabilidade entre as paredes. O modelo de visibilidade não admite nenhuma permeabilidade e por causa disso acaba tendo que contornar o obstáculo. A existência ou não dessa permeabilidade é extremamente dependente do ambiente e do material do qual as paredes são feitas. Além disso, o modelo de visibilidade consegue fornecer uma informação que o modelo radial não consegue. O modelo de visibilidade consegue garantir que um determinado trajeto é completamente realizável. Isso quer dizer que o modelo de visibilidade consegue garantir que o trajeto completo até a tarefa pode ser percorrido sem perda de conexão com aquele número de robôs. Para melhor visualizar isso, imagine que o robô mais próximo da tarefa na Figura 3.14a seja o robô "A" e que essa nomenclatura se estenda até o robô "D", dentro da zona de conectividade. Se os quatro robôs começarem dentro da zona de conectividade na posição do robô "D", e todos menos o robô "D" começarem a se deslocar para a posição "C", eles todos estarão conectados por meio do robô "D". Ao chegar na posição "C", os robôs "A" e "B" continuam o trajeto até o ponto "B", enquanto o robô "C" fica para trás para garantir a conexão desses dois, e assim por diante. O cenário com dois robôs da conexão radial, por mais que utilize menos robôs, não possui nenhuma garantia de ser completamente realizável. O algoritmo necessário para verificar isso seria mais custoso, e nesse caso específico, retornaria que não é possível realizar essa tarefa com apenas dois robôs, sem que eles percam conexão durante o trajeto. Por esse motivo, foi optado por se utilizar o modelo de conexão por visibilidade no planejamento de missão.

Com o modelo de conexão escolhido, pode-se seguir para a segunda parte do algoritmo de busca, a fase de busca. Com a geração do mapa de busca $\mathcal{M}(V, E, D)$, o Algoritmo 3.3 pode ser utilizado para calcular o conjunto $\mathbf{T}_\alpha$ associado a uma tarefa $t \in \mathbf{T}$. Essas tarefas-α são correspondentes as posições dos robôs auxiliares "B", "C" e "D" na Figura 3.14a.

Se t se encontra dentro da zona de conectividade, ou seja, $\mathfrak{C}(t_p) \geq \gamma_{\mathfrak{C}}$, nenhuma tarefa-$\alpha$ será necessária, e apenas um conjunto contendo a tarefa t é retornado. Caso contrário, a função CAMINHO_CONEXÃO_-MAIS_PRÓXIMA utiliza o mapa de busca $\mathcal{M}$ para encontrar o menor caminho até a zona de conectividade.

Na implementação realizada, isso é feito por meio da busca do caminho de menor resistência até uma amostra de custo zero no mapa de busca. O primeiro passo é conectar t a alguma amostra já existente em

Algoritmo 3.3 Calcular tarefas-α

1: **função** CALCULAR_TAREFAS_ALFA($\mathfrak{C}$, $\mathcal{M}$, t)
2: **se** $\mathfrak{C}(t_p) \geq \gamma_{\mathfrak{C}}$ **então**
3: **retorna** $\{t\}$
4: **fim se**
5: $\mathbf{T}'_\alpha \leftarrow$ CAMINHO_CONEXÃO_MAIS_PRÓXIMA $(\mathcal{M}, t, \mathfrak{C})$
6: $\mathbf{T}_\alpha \leftarrow$ SIMPLIFICAR $(\mathfrak{C}, \mathbf{T}'_\alpha)$
7: **retorna** $\mathbf{T}_\alpha$
8: **fim função**

$\mathcal{M}$. Essa amostra deve conter o menor custo total até a zona de conectividade. Para isso, uma função $h_{\mathfrak{C}} : \mathcal{W} \times \mathcal{W} \rightarrow \mathbb{R}^+$ é definida como o custo em linha reta entre a posição da tarefa t_p e a posição de um vértice $v \in V$, se esse caminho for livre de colisões. Caso exista alguma colisão o valor dessa função é ∞. Assim, basta encontrar a amostra $v_{min} \in V$ que possua o menor valor para

$$h_{\mathfrak{C}}(t_p, v) + \mathcal{D}(v) \tag{3.7}$$

entre todas as amostras de V.

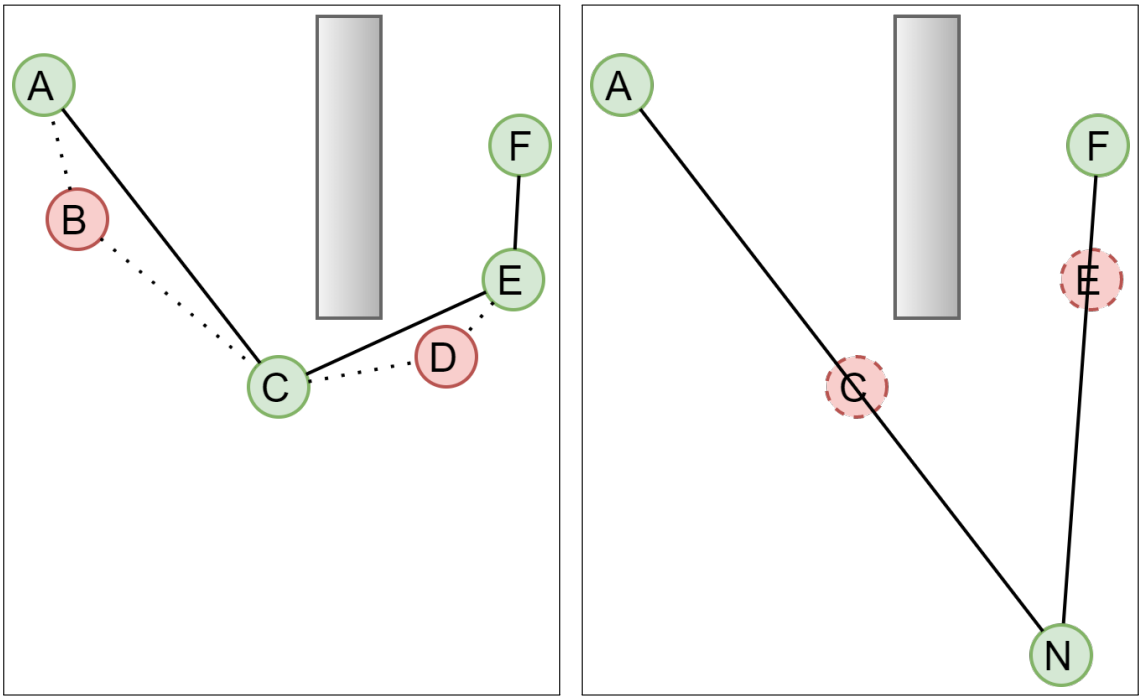
Assim que essa amostra v_{min} é encontrada, o próximo passo é procurar por outra amostra v_a que seja conectada a ela por uma aresta $e \in E$ que contenha o menor valor $\mathcal{D}(v_a)$ possível. E então, repetir isso com cada amostra, até que uma amostra $\mathcal{D}(v_a) = 0$ seja encontrada. As amostras com esse valor se encontram dentro da zona de conectividade. O conjunto de amostras v_a que foram utilizadas para sair da tarefa t, incluindo t, até a amostra em que $\mathcal{D}(v_a) = 0$ formando o conjunto $\mathbf{T}'_\alpha$. Esse conjunto é o conjunto base do qual as tarefas-α de t serão extraídas.

Se $D(v_{min}) = \infty$, então não existe nenhum caminho válido que ligue a tarefa t até a região de conexão. Quando isso acontece a função CAMINHO_CONEXÃO_MAIS_PRÓXIMA retorna o conjunto vazio $\emptyset$, representando que essa tarefa não é alcançável pelo mapa de busca atual.

A transformação do conjunto base $\mathbf{T}'_\alpha$ para o conjunto final $\mathbf{T}_\alpha$ é feita pela função SIMPLIFICAR, para isso, ela realiza duas operações. A Figura 3.15 mostra uma representação desses dois passos. Os pontos de A até F representam as amostras do conjunto T'_α. Considerando que o ponto A é a tarefa t e que o ponto F é a amostra que pertence a zona de conectividade.

O primeiro passo (Figura 3.15a) é a remoção de pontos redundantes, como por exemplo, os ponto B e D. Considerando três amostras x, y e z que estejam conectas pelas arestas $\overline{xy}$ e $\overline{yz}$, a amostra y é chamada redundante se a aresta $\overline{xz}$ pode ser construída sem colidir com nenhum obstáculo. Por exemplo, o ponto C não é redundante, pois se ele for removido o caminho não conseguirá se manter conectado.

O segundo passo (Figura 3.15b) é tentar criar novos pontos que consigam reduzir o número total de pontos no caminho. Isso geralmente aumenta o tamanho total do caminho, mas nesse caso o número de pontos é mais importante que o tamanho do caminho. Para tentar criar esses pontos é calculada a intersecção das retas que contenham um segmento de reta formado por duas amostras consecutivas. Caso o novo segmento de reta formado entre os pontos originais e a intersecção não possua nenhuma colisão, então o número de pontos do caminho pode ser reduzido. Por exemplo, na Figura 3.15b, as retas que contêm os segmentos $\overline{AC}$ e $\overline{EF}$ se interseccionam no ponto N. Como os novos segmentos $\overline{AN}$ e $\overline{NF}$ não colidem com nenhum obstáculo, é possível remover os pontos C e E e adicionar o ponto N ao conjunto $\mathbf{T}'_\alpha$ gerando o conjunto $\mathbf{T}_\alpha$. Dessa



(a) Remoção de pontos redundantes.

(b) Criação de pontos substitutos.

Figura 3.15: Simplificação do conjunto conjunto base $\mathbf{T}'_\alpha$ para o conjunto final $\mathbf{T}_\alpha$

maneira, o conjunto final $\mathbf{T}_\alpha$ é $\mathbf{T}_\alpha = \{A, N, F\}$, mostrando que três robôs são capazes de realizar a tarefa t localizada no ponto A sem perder conexão.

A existência e o número de tarefas-α determina a completude da tarefa t associada a elas. Caso uma tarefa t, fora da zona de conectividade, não possua tarefas-α, t é considerada não realizável. Considerando que $\mathcal{N}_\alpha(t) = |\mathbf{T}_\alpha| - 1$ é o número de tarefas-α associadas a t, então o número de robôs necessários para realizar uma tarefa é $\mathcal{N}_\alpha(t) + 1$. Isso representa todos os robôs sendo utilizados como extensores de rede e o robô executando de fato a tarefa. Por exemplo, no cenário da Figura 3.14a são necessários pelo menos quatro robôs para realizar a tarefa desejada.

3.4 DECIDINDO QUAL TAREFA EXECUTAR

A decisão de qual tarefa executar necessita levar alguns fatores em consideração, como a prioridade m_π, o prazo máximo de execução $m_\mathfrak{d}$, a recompensa $\mathbf{r}$ e o esforço total necessário para realizar a mesma. O esforço total não depende apenas da função de esforço $\mathbf{e}$, pois é necessário que o robô se desloque da sua posição atual até a tarefa t. Para completar uma missão m é necessário somar o esforço de execução de todas as tarefas t_k, que pertençam a $m_\mathbf{T}$, com o esforço de cada uma das tarefas pré requisitos $t^k_\mathbf{T}$, e assim sucessivamente até que todas as tarefas pré requisitos tenham sido consideradas. A Figura 3.16 exemplifica isso, exibindo toda a cadeia de dependência de uma missão m. Os retângulos pontilhados verdes representam o conjunto $m_\mathbf{T} = \{t_8, t_9\}$ que é o conjunto das tarefas que fazem parte da missão diretamente. Os retângulos pontilhados azuis representam o conjunto de tarefas pré requisitos $t^i_\mathbf{T}$. Como os conjuntos $t^6_\mathbf{T}$ e $t^7_\mathbf{T}$ são iguais $t^6_\mathbf{T} = t^7_\mathbf{T} = \{t_4\}$, eles são

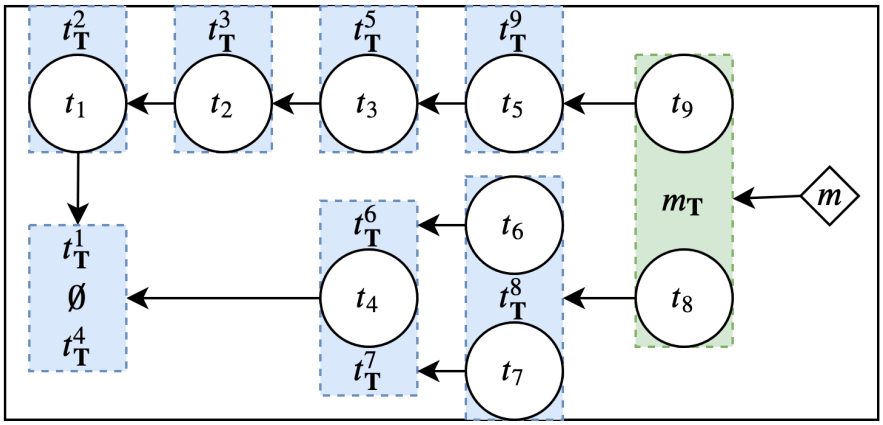


Figura 3.16: Todas as tarefas que precisam ser realizadas para que a missão m esteja completa.

representados pelo mesmo retângulo. O mesmo ocorre para $t^1_{\mathbf{T}}$ e $t^4_{\mathbf{T}}$ que não possuem pré requisitos. Sendo assim, o esforço completo para completar a missão m é a soma das funções de esforço e para todas as tarefas de t_1 até t_9, mais o deslocamento entre essas tarefas.

Em um cenário em que a manutenção de comunicação não é fundamental, o esforço de deslocamento pode ser medido pelo comprimento do caminho que será percorrido dividido pela velocidade média do robô. Existem algoritmos capazes de calcular essas rotas rapidamente e obter um cálculo preciso, no entanto, em alguns casos é necessário sacrificar a precisão para se ganhar velocidade no cálculo. Para isso são utilizadas heurísticas que consigam fazer uma suposição sobre qual seria o tamanho da rota necessária. Por exemplo, em espaços euclidianos uma heurística muito utilizada é a distância euclidiana. Ela não considera a presença de obstáculos, mas é muito mais veloz para ser computada em relação a um cálculo de rota.

Sacrificando um pouco de velocidade de cálculo, é possível melhorar um pouco uma heurística, por exemplo, unindo a distância euclidiana a uma varredura em linha reta entre os dois pontos, permite que uma penalidade seja adicionada cada vez que um obstáculo seja encontrado. Essa heurística é mais complexa que simplesmente calcular a distância, porém menos complexa do que calcular a rota completa, e com isso consegue incorporar mais informações na suposição que está sendo feita.

Na aplicação apresentada, o cálculo de um rota livre de colisões entre dois pontos é realizada rapidamente por meio de um algoritmo de busca bidirecional com RRT* (Seção 2.4.5). Sendo assim, caso o robô e a tarefa que ele deseje realizar se encontrem totalmente dentro de uma zona de conectividade contínua, o esforço de deslocamento é simplesmente computado pelo tamanho dessa rota. Quando um dos dois não está dentro da zona de conectividade, esse planejamento se torna muito mais complexo, pois o robô não pode simplesmente se mover livremente por uma zona sem conectividade, ele precisa de um, ou mais robôs, que o auxilie nessa locomoção. As tarefas-α permitem obter uma maior clareza sobre o esforço coletivo necessário para realizar uma determinada tarefa t fora da zona de conectividade sem perda de conexão. Sendo assim, é possível utilizar as tarefas-α para gerar heurísticas que tentem refletir esse esforço.

Como explicado anteriormente, podem existir múltiplas heurísticas para o mesmo problema, e, geralmente, quão mais complexas forem essas heurísticas, mais informações elas irão agregar, entretanto, elas serão mais complexas de se calcular. Por causa disso, algumas heurísticas diferentes serão propostas nessa etapa, de forma que uma análise comparativa possa ser feita entre elas. Essas heurísticas obedecem duas principais formas

$$h_1(t) = f(\mathbf{T}_\alpha)^{|\mathbf{T}_\alpha|} \tag{3.8}$$

e

$$h_2(t) = \sum_{i=1}^{|\mathbf{T}_\alpha|} g(r^i_{\alpha t}, \alpha_i t). \tag{3.9}$$

As heurísticas na forma da Equação 3.9, aqui chamadas de heurísticas h_2, representam a soma de heurísticas para robôs alcançarem as tarefas pertencentes ao conjunto $\mathbf{T}_\alpha$ de uma tarefa t calculado na etapa anterior. Nesse caso, $|\mathbf{T}_\alpha|$ representa o número de elementos no conjunto $\mathbf{T}_\alpha$; $\alpha_i t$ representam as tarefas desse conjunto, com $\alpha_{|\mathbf{T}_\alpha|} t$ sendo a própria tarefa t; $r^i_{\alpha t} \in \mathbf{R}_t$ é o robô responsável por realizar a i-ésima tarefa-α, em que $\mathbf{R}_t \subset \mathbf{R}$ é um subconjunto dos robôs utilizados para realizar a missão t; e g é uma função heurística para estimar a quantidade de esforço que o robô $r^i_{\alpha t}$ fará para realizar a tarefa $\alpha_i t$.

Já as heurísticas no formato da Equação 3.8, heurísticas h_1, são formas mais simples, elas não levam em consideração o posicionamento dos robôs e qual robô deve fazer qual tarefa. Para compensar isso, elas elevam suas estimativas ao número de tarefa-α mais uma, dando um peso maior para tarefas t que possuam muitas tarefas-α. A função f seria uma heurística sobre todas as tarefas do conjunto $\mathbf{T}_\alpha$.

Em um cenário onde existem $|\mathbf{R}|$ robôs e $|\mathbf{T}_\alpha|$ tarefas, considerando que t é a $|\mathbf{T}_\alpha|$-ésima tarefa-α, existem $\binom{|\mathbf{R}|}{|\mathbf{T}_\alpha|}$ possibilidades diferentes de organizar os robôs para realizar essas tarefas. Em cada uma dessas possibilidades existem $|\mathbf{T}_\alpha|!$ maneiras diferentes de realizar essas tarefas. Sendo assim, existem no total

$$\begin{aligned} n_p &= \binom{|\mathbf{R}|}{|\mathbf{T}_\alpha|} |\mathbf{T}_\alpha|!, & (3.10)\\ n_p &= \frac{|\mathbf{R}|!}{|\mathbf{T}_\alpha|!\,(|\mathbf{R}| - |\mathbf{T}_\alpha|)!} |\mathbf{T}_\alpha|!, & (3.11)\\ n_p &= \frac{|\mathbf{R}|!}{(|\mathbf{R}| - |\mathbf{T}_\alpha|)!} & (3.12) \end{aligned}$$

maneiras diferentes de se executar essas tarefas-α $\mathbf{T}_\alpha$ com os robôs do conjunto $\mathbf{R}$. Isso representa o número de casos das funções do tipo h_2 que precisam ser verificados para determinar o menor custo estimado. As funções heurísticas h_1 são mais simples justamente para evitar essa necessidade de verificar n_p possibilidades.

As funções heurísticas h_1 inicialmente propostas foram as seguintes:

$$\begin{aligned} h_{1a}(t) &= \left(\sum_{i=1}^{|\mathbf{T}_\alpha|-1} \rho(\alpha_i t, \alpha_{i+1} t) \right)^{|\mathbf{T}_\alpha|}; & (3.13)\\ h_{1b}(t) &= \left(\Phi(\mathbf{T}_\alpha)\right)^{|\mathbf{T}_\alpha|}. & (3.14) \end{aligned}$$

Elas representam extensões de dois casos nos quais não existe uma preocupação com a comunicação. A primeira h_{1a}, Equação 3.13, representa a distância euclidiana ρ entre as tarefas. A ideia é somar a distância em linha reta entre essas tarefas e elevar isso ao número de tarefas, tentando super estimar o esforço extra decorrente de movimentar vários robôs e manter a conexão. A segunda h_{1b}, Equação 3.14, possui exatamente a mesma lógica, a única diferença é que a função Φ representa o caminho entre as tarefas desviando de obstáculos. Após

uma análise, percebeu-se que ambas h_{1a} e h_{1b} são de certa forma a mesma função, pois devido a natureza das tarefas-α o caminho em linha reta entre duas tarefas-α consecutivas não colide com nenhum obstáculo. A função h_{1b} permite caminhos curvos e outras configurações que a função h_{1a} não permite, mas como se busca pelo custo mínimo, h_{1a} representa a forma ótima de h_{1b} e, por esse motivo, a partir daqui será considerada apenas a função h_{1a} para análises mais complexas.

As funções heurísticas h_2 propostas foram:

$$h_{2a}(t) = \sum_{i=1}^{|\mathbf{T}_\alpha|} \rho_{\pi k}(r^i_{\alpha t}, \alpha_i t); \tag{3.15}$$

$$h_{2b}(t) = \sum_{i=1}^{|\mathbf{T}_\alpha|} \Phi_{\pi k}(r^i_{\alpha t}, \alpha_i t); \tag{3.16}$$

$$h_{2c}(t) = \sum_{i=1}^{|\mathbf{T}_\alpha|} \frac{\rho(r^i_{\alpha t}, \alpha_i t)}{\Psi(r^i_{\alpha t}, \alpha_i t)}; \tag{3.17}$$

$$h_{2d}(t) = \sum_{i=1}^{|\mathbf{T}_\alpha|} \frac{\Phi(r^i_{\alpha t}, \alpha_i t)}{\Psi(r^i_{\alpha t}, \alpha_i t)}. \tag{3.18}$$

Essa funções necessitam que seja decidido qual robô realizará qual tarefa, para isso essas funções devem ser resolvidas de maneira que

$$h_2(t) = \min_{\mathbf{R}_t} \sum_{i=1}^{|\mathbf{T}_\alpha|} g(r^i_{\alpha t}, \alpha_i t), \tag{3.19}$$

em que $\mathbf{R}_t$ é o conjunto que representa qual robô executa cada tarefa de $\mathbf{T}_\alpha$. Como explicado, isso cresce de acordo com n_p, Equação 3.12, e pode ficar bem complexo para um número muito alto de tarefas e robôs. A função h_{2a}, Equação 3.15, representa na verdade uma família de equações $\rho_{\pi k}$. A função $\rho_{\pi k}$ calcula a distância euclidiana entre dois pontos e multiplica por uma constante k os trechos fora da zona de conectividade. A função h_{2b}, Equação 3.16, possui a mesma lógica, apenas trocando a distância euclidiana $\rho_{\pi k}$ pela distância do caminho sem colisão $\Phi_{\pi k}$ entre o robô e a tarefa que ele irá realizar. As funções h_{2c} e h_{2d}, respectivamente Equações 3.17 e 3.18, utilizam a distância euclidiana e a distância sem colisão e dividem ela pela função Ψ, que representa a porcentagem dessa distância que se encontra dentro da zona de conectividade.

Independentemente de qual heurística se utiliza, o algoritmo para decidir qual missão será executada é o mesmo. O importante é que após analisar uma tarefa com uma dada heurística, todas as outras tarefas devem ser analisadas com a mesma heurística. Outro fator importante a se considerar é o fato de que uma nova missão pode chegar a qualquer momento. Isso significa que não faz sentido calcular um planejamento ótimo de todas as missões disponíveis, pois após a realização de uma missão, pode ser que novas missões tenham chegado, tornando inútil o planejamento anterior. Dessa maneira, uma estratégia mais gulosa foi utilizada, ou seja, apenas a missão que atender mais critérios e der a maior recompensa será realizada.

A missão a ser realizada é definida de acordo com os passos a seguir. Inicialmente se escolhem as missões que formam o conjunto da missões de maior prioridade $\mathbf{M}_\pi \subset \mathbf{M}$, esse conjunto possui apenas as missões que possuam o menor valor possível em m_π de todas as missões em $\mathbf{M}$. Caso apenas uma missão exista em $\mathbf{M}_\pi$, essa será a missão a ser executada. Caso contrário, o próximo item a se analisar é o prazo máximo das funções $m_\mathfrak{d}$. A missão escolhida é a que possuir menor folga de realização. Essa folga $\mathfrak{f}$ é caracterizada pela diferença

entre o prazo máximo e o esforço total necessário para realizar essa missão, ou seja,

$$\mathfrak{f}(m) = m_{\mathfrak{d}} - \sum_{i=1}^{N} \left(h\left(t_i\right) + \mathbf{e}\left(t_\tau^i, r_\tau^i\right)\right), \tag{3.20}$$

em que N é o número de tarefas que precisam ser realizadas para considerar a missão m completa; h é uma das funções heurísticas apresentadas; $\mathbf{e} : \mathcal{T}_\mathbf{T} \times \mathcal{T}_\mathcal{R} \rightarrow \mathbb{R}^+$ é a função de esforço; t_i é uma das tarefas que precisam ser realizadas para completar m; e r_τ^i é o tipo do robô responsável por executar a tarefa t_i. Por fim, caso exista um empate de folga a função de recompensa $\mathbf{r} : \mathcal{T}_\mathbf{T} \rightarrow \mathbb{R}^+$ é utilizada como critério de desempate da seguinte maneira: a missão que possuir a maior recompensa de missão $\mathbf{r}_m$ é escolhida, em que

$$\mathbf{r}_m(m) = \sum_{i=1}^{N} \left(\mathbf{r}\left(t_\tau^i\right)\right), \tag{3.21}$$

ou seja, a recompensa de uma missão é igual a soma da recompensa de suas tarefas. Seguindo esses passos é possível decidir, de maneira gulosa, qual tarefa deve ser executada primeiro.

3.5 CRIANDO UM PLANO DE EXECUÇÃO

A criação de um plano de execução completo de uma tarefa t pode ser algo muito complexo quando é necessário considerar que o robô não pode perder sua comunicação durante o seu deslocamento. Por causa disso, os dois passos anteriores foram considerados fundamentais. A criação de tarefas-α direciona a busca por um plano de execução, enquanto que a escolha de qual tarefa deve ser executada por meio de heurísticas, permite que um plano seja computado apenas para a missão m escolhida. Diminuindo a quantidade de processamento.

Para gerar um plano de execução de uma determinada missão m, é necessário gerar os plano de execução de todas as tarefas que precisam ser executadas para completar essa missão. A etapa anterior define quais robôs vão executar cada uma das tarefas t e quais robôs vão executar cada uma das tarefas-α associadas a elas. Com todos esses parâmetros definidos, é possível utilizar um algoritmo de busca em árvore, Seção 2.4.5 , para gerar um plano de execução de uma missão t por um conjunto de robôs $\mathbf{R}_t \subset \mathbf{R}$.

O primeiro passo para gerar esse plano é definir o espaço de configuração sobre o qual será trabalhado. De acordo com a Equação 2.4, em um problema com múltiplos robôs, o espaço de configuração de planejamento total é o produto cartesiano dos espaços de configurações individuais. Para uma melhor compreensão, o resto do desenvolvimento será feito considerando robôs omnidirecionais se deslocando em um ambiente bidimensional. Sendo assim, se existirem $|\mathbf{R}_t|$ robôs para realizar uma tarefa, o espaço de configuração $\mathcal{C}$ será

$$\mathcal{C} = \mathbb{R}^{2|\mathbf{R}_t|}. \tag{3.22}$$

Considerando que uma amostra $x \in \mathbb{R}^{2|\mathbf{R}_t|}$, possa ser rescrita como

$$x = \{x_1, x_2, \ldots, x_{|\mathbf{R}_t|}\}, \tag{3.23}$$

em que $x_i \in \mathbb{R}^2$ corresponde a posição do i-ésimo robô, a função de distância $\rho_{2|\mathbf{R}_t|}$ entre duas amostras $a, b \in \mathcal{C}$ pode ser definida como

$$\rho_{2|\mathbf{R}_t|}(a,b) = \sum_{i=1}^{|\mathbf{R}_t|} \rho_2(a_i, b_i), \tag{3.24}$$

sendo que $\rho_2 : \mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{R}^+$ é a distância euclidiana bidimensional, a_i e b_i são as posições do i-ésimo robô nas configurações a e b. Ou seja, a distância é medida pela soma das distâncias individuais entre as posições do mesmo robô. Para que essa medida seja um função de distância válida, ela deve obedecer os quatro requisitos descritos na Seção 2.4.5.

O primeiro requisito é que o valor de $\rho_{2|\mathbf{R}_t|}(a,b)$ deve ser não negativo, finito e real. A função $\rho_2(a_i, b_i)$ obedece esses requisitos, e como $\rho_{2|\mathbf{R}_t|}(a,b)$ é apenas uma soma dessa primeira, a soma de números não negativos, finitos e reais também é um número não negativo, finito e real. Atendendo, assim, o primeiro requisito.

O segundo requisito é a reflexividade, ou seja $\rho_{2|\mathbf{R}_t|}(a,b) = 0$ se e somente se $a = b$. A ida desse requisito é fácil de provar, pois

$$a = b \iff a_i = b_i \forall i \in \{1, 2, \ldots, |\mathbf{R}_t|\} \tag{3.25}$$

então se $a = b$

$$\rho_{2|\mathbf{R}_t|}(a,b) = \sum_{i=1}^{|\mathbf{R}_t|} \rho_2(a_i, b_i) = \sum_{i=1}^{|\mathbf{R}_t|} 0 = 0, \tag{3.26}$$

pois ρ_2 é reflexivo, fazendo com que $\rho_2(a_i, b_i) = 0$ se $a_i = b_i$. A volta é provada partindo desse ponto. Como ρ_2 é reflexivo, a única maneira de $\rho_2(a_i, b_i) = 0$ é se $a_i = b_i$. Como $\rho_{2|\mathbf{R}_t|}$ é uma soma de números não negativos, a única maneira de $\rho_{2|\mathbf{R}_t|}(a,b) = 0$ é se todos os termos dessa soma forem zero. Sendo assim, pela reflexividade $a_i = b_i \forall i \in \{1, 2, \ldots, |\mathbf{R}_t|\}$, o que implica que $a = b$. Provando que $\rho_{2|\mathbf{R}_t|}(a,b)$ é reflexivo.

O terceiro requisito é a simetria, ou seja, $\rho_{2|\mathbf{R}_t|}(a,b) = \rho_{2|\mathbf{R}_t|}(b,a)$. Essa é bem simples de provar considerando que $\rho_2(a_i, b_i) = \rho_2(b_i, a_i)$, pois a função ρ_2 é simétrica. Sendo assim,

$$\rho_{2|\mathbf{R}_t|}(b,a) = \sum_{i=1}^{|\mathbf{R}_t|} \rho_2(b_i, a_i), \tag{3.27}$$

$$\rho_{2|\mathbf{R}_t|}(b,a) = \sum_{i=1}^{|\mathbf{R}_t|} \rho_2(a_i, b_i), \tag{3.28}$$

$$\rho_{2|\mathbf{R}_t|}(b,a) = \rho_{2|\mathbf{R}_t|}(a,b). \tag{3.29}$$

Provando, assim, a sua simetria.

Por último, tem-se o requisito de desigualdade triangular. Considerando um terceiro ponto $c \in \mathcal{C}$ deve valer a seguinte desigualdade

$$\rho_{2|\mathbf{R}_t|}(a,b) + \rho_{2|\mathbf{R}_t|}(b,c) \geq \rho_{2|\mathbf{R}_t|}(a,c). \tag{3.30}$$

Para provar isso, a inequação 3.30 será expandida a seguir.

$$\rho_{2|\mathbf{R}_t|}(a,b) + \rho_{2|\mathbf{R}_t|}(b,c) \geq \rho_{2|\mathbf{R}_t|}(a,c) \tag{3.31}$$

$$\sum_{i=1}^{|\mathbf{R}_t|} \rho_2(a_i,b_i) + \sum_{i=1}^{|\mathbf{R}_t|} \rho_2(b_i,c_i) \geq \sum_{i=1}^{|\mathbf{R}_t|} \rho_2(a_i,c_i) \tag{3.32}$$

$$\sum_{i=1}^{|\mathbf{R}_t|} [\rho_2(a_i,b_i) + \rho_2(b_i,c_i)] \geq \sum_{i=1}^{|\mathbf{R}_t|} \rho_2(a_i,c_i) \tag{3.33}$$

Como a desigualdade triangular é valida para ρ_2,

$$\rho_2(a_i,b_i) + \rho_2(b_i,c_i) \geq \rho_2(a_i,c_i). \tag{3.34}$$

Sendo assim, a inequação 3.33 tem para cada i-ésimo termo de sua soma do lado direito, um i-ésimo termo de tamanho maior ou igual ao seu do lado esquerdo, fazendo com que a soma total do lado esquerdo seja de fato maior ou igual a soma total do lado direito, provando assim que a desigualdade triangular é válida para a função $\rho_{2|\mathbf{R}_t|}$. Dessa maneira, $\rho_{2|\mathbf{R}_t|}$ cumpre os quatro requisitos necessários e pode ser utilizada como uma função de distância para o espaço $\mathbb{R}^{2|\mathbf{R}_t|}$.

A amostragem de uma amostra $x \in \mathbb{R}$ pode ser feita a partir da função de amostragem bidimensional $\alpha_2 : \mathbb{N} \to \mathbb{R}^2$. Basta definir uma função de amostragem $\alpha_{2|\mathbf{R}_t|} : \mathbb{N} \to \mathbb{R}^{2|\mathbf{R}_t|}$ que utilize a função bidimensional da seguinte maneira para a i-ésima amostra,

$$x = \alpha_{2|\mathbf{R}_t|}(i) = \underbrace{\{\alpha_2(|\mathbf{R}_t|\, i,)\, \alpha_2(|\mathbf{R}_t|\, i+1), \ldots, \alpha_2(|\mathbf{R}_t|\, i + |\mathbf{R}_t| - 1)\}}_{|\mathbf{R}_t| \text{ amostras}}, \tag{3.35}$$

ou seja, a posição de cada j-ésimo robô x_j é dada pela amostra bidimensional de índice $|\mathbf{R}_t|\, i + (j-1)$.

A detecção de colisão á baseada na detecção de colisão do cálculo de tarefas-α. A área obstruída $\mathcal{W}_O$ e a função de conectividade $\mathfrak{C}$ são utilizadas para construir o espaço $\mathcal{C}_{obs}$ de maneira implícita. Para facilitar a função $\mathcal{O}(q) : \mathcal{W} \to \{0,1\}$ será utilizada para dizer se um robô q, que representa a posição de um dos $|\mathbf{R}_t|$ robôs de uma amostra $x \in \mathcal{C}$, possui ou não uma colisão com um obstáculo presente em $\mathcal{W}_O$.

Usando a notação da Equação 3.23, uma configuração $x \in \mathcal{C}$ pode ser escrita em função da posição dos seus $|\mathbf{R}_t|$ robôs, como representado na Figura 3.17a. Uma colisão nesse espaço de configuração ocorre em algumas diferentes situações. A primeira e mais simples delas é que nenhum de seus robôs pode colidir com um obstáculo, ou seja, existe colisão se

$$\exists x_i \in x | \mathcal{O}(x_i) = 1. \tag{3.36}$$

A segunda situação é que todos os robôs não podem estar fora da zona de conectividade, ou seja, existe colisão se

$$\forall x_i \in x, \mathfrak{C}(x_i) \leq \gamma_{\mathfrak{C}}. \tag{3.37}$$

Caso nenhuma das duas situações acima ocorra, existem dois possíveis cenários. O primeiro é quando

$$\forall x_i \in x, \mathfrak{C}(x_i) > \gamma_{\mathfrak{C}}, \tag{3.38}$$

ou seja, todas os robôs não colidem com nenhum obstáculo físico e estão dentro da zona de conectividade. Nesse cenário não existe colisão. O segundo cenário dessa última situação é que existem alguns robôs dentro da zona de conectividade e alguns fora. Nesse cenário, existe colisão se existir algum robô que não tenha visibilidade a nenhum outro robô que esteja dentro da zona de conectividade, ou que não veja nenhum robô que faça parte de uma cadeia de conexão até a zona de conectividade.

Entretanto, detectar se uma única configuração pertence a $\mathcal{C}_{obs}$ não é o bastante. A função de detecção de colisão deve ser capaz de verificar se existe alguma colisão na aresta formada entre duas configurações $x, y \in \mathcal{C}$. Considerando que

$$x = \{x_1, x_2, \ldots, x_{|\mathbf{R}_t|}\} \text{ e} \tag{3.39}$$

$$y = \{y_1, y_2, \ldots, y_{|\mathbf{R}_t|}\}. \tag{3.40}$$

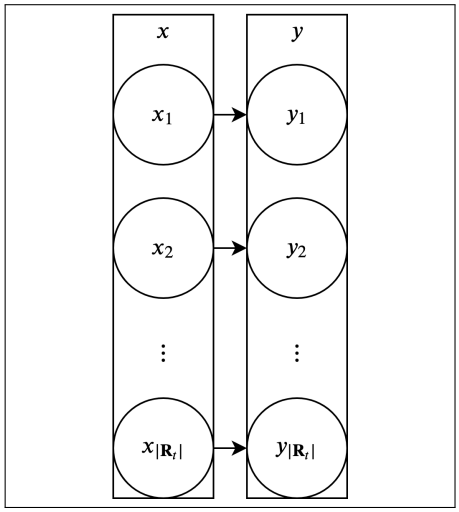
Então, uma aresta e entre as configurações x e y pode ser visualizada como uma aresta entre cada um dos robôs que fazem parte dessas configurações, em outras palavras, uma aresta entre x e y corresponde a $|\mathbf{R}_t|$ arestas entre robôs no formato $e_i = \{x_i, y_i\}$. Para averiguar se uma aresta possui alguma colisão, cada uma das e_i é amostrada em intervalos δ_{obs}. Esse intervalo deve corresponder ao tamanho do menor obstáculo, ou ao tamanho do menor intervalo onde poderia ocorrer um quebra de conexão. Como as arestas entre os robôs possuem tamanhos diferentes, esses caminhos terão um número diferentes de amostras. Considerando que

$$\boldsymbol{e}_i = y_i - x_i \tag{3.41}$$

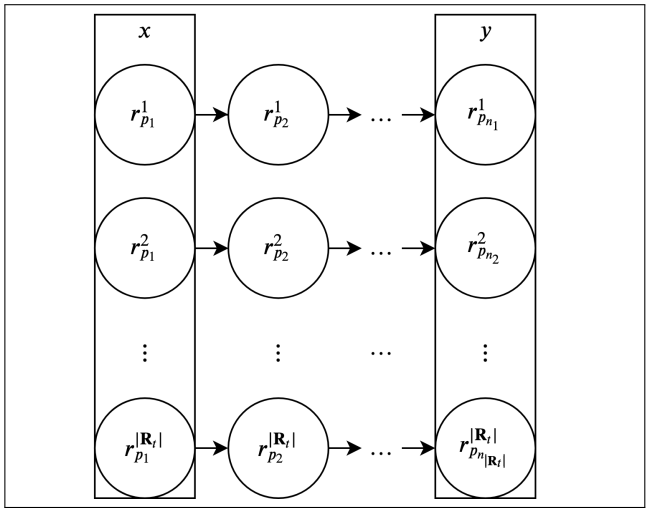
é o vetor que representa a aresta e_i, então o caminho formado por essa aresta possui

$$n_i = \left\lceil \frac{\|\boldsymbol{e}_i\|}{\delta_{obs}} \right\rceil \tag{3.42}$$

pontos $r^i_{p_j}$, com $j \in \{1, 2, \ldots, n_i\}$. Em que $\|\boldsymbol{e}_i\|$ é a norma do vetor $\boldsymbol{e}_i$ e $\lceil\rceil$ arredonda um número para cima. Sendo assim, as configurações x e y podem ser reescritas como



(a) Representação gráfica de uma aresta entre duas configurações $x, y \in \mathcal{C}$.



(b) Representação gráfica da amostragem da aresta e entre duas configurações $x, y \in \mathcal{C}$.

Figura 3.17: Visualizando arestas em $\mathcal{C}$.

$$x = \{r^1_{p_1}, r^2_{p_1}, \ldots, r^{|\mathbf{R}_t|}_{p_1}\} \text{ e} \tag{3.43}$$
$$y = \{r^1_{p_{n_1}}, r^2_{p_{n_2}}, \ldots, r^{|\mathbf{R}_t|}_{p_{n_{|\mathbf{R}_t|}}}\}. \tag{3.44}$$

Essa nova representação de x e y, assim como as amostras entre eles, está representada na Figura 3.17b. Todas essas configurações formam o conjunto $V_{\mathfrak{C}}$.

Esse conjunto $V_{\mathfrak{C}}$ será a base sobre a qual os grafo de visibilidade e de não visibilidade serão construídos. Esses grafos serão utilizados para determinar se existe ou não colisão na aresta e entre x e y. Para simplificar a notação, a j-ésima posição do i-ésimo robô $r^i_{p_j}$, será denotada por r^i_j a partir desse momento. O primeiro passo é verificar para cada amostra $r^i_j \in V_{\mathfrak{C}}$ se $\mathcal{O}(r^i_j) = 0$, caso exista pelo menos uma amostra em que isso não é verdade, existe colisão e a verificação pode parar por aqui. Caso nenhuma colisão física ocorra, o próximo passo é marcar todas as amostras $r^i_j \in V_{\mathfrak{C}}$ que estejam na zona de conectividade, ou seja, $\mathfrak{C}(r^i_j) > \gamma_{\mathfrak{C}}$, como representado na Figura 3.18, que é um exemplo de marcação para o exemplo da Figura 3.19. Caso não exista pelo menos um robô i que possua seu caminho completo dentro da zona de conectividade, como por exemplo o robô 3 na Figura 3.18, então é impossível ir da amostra x para y sem que falhe a a comunicação e portanto, é considerado que existe colisão. Outro cenário fácil de verificar é se todas as amostras $r^i_j \in V_{\mathfrak{C}}$ estão dentro da zona de conectividade, nesse caso, é possível ir de x para y sem bater em obstáculos e mantendo uma conexão, e não, portanto, é considerado que não existe colisão.

Caso nenhuma das condições de saída tenha sido alcançada até agora, o próximo passo seria verificar se existe alguma maneira de realizar esses trajetos mantendo uma conexão. No entanto, antes disso, existe mais uma condição rápida de se verificar que pode gerar uma colisão. Considerando que a configuração x é sempre válida, o trajeto entre x e y é impossível se a configuração y for impossível. Portanto, o próximo passo será verificar se a configuração y é possível. Essa etapa também será utilizada para começar a construção dos grafos de visibilidade $\mathcal{G}_{\text{vis}}(V_{\mathfrak{C}}, E_{\text{vis}})$ e não visibilidade $\mathcal{G}_{\not{\text{vis}}}(V_{\mathfrak{C}}, E_{\not{\text{vis}}})$.

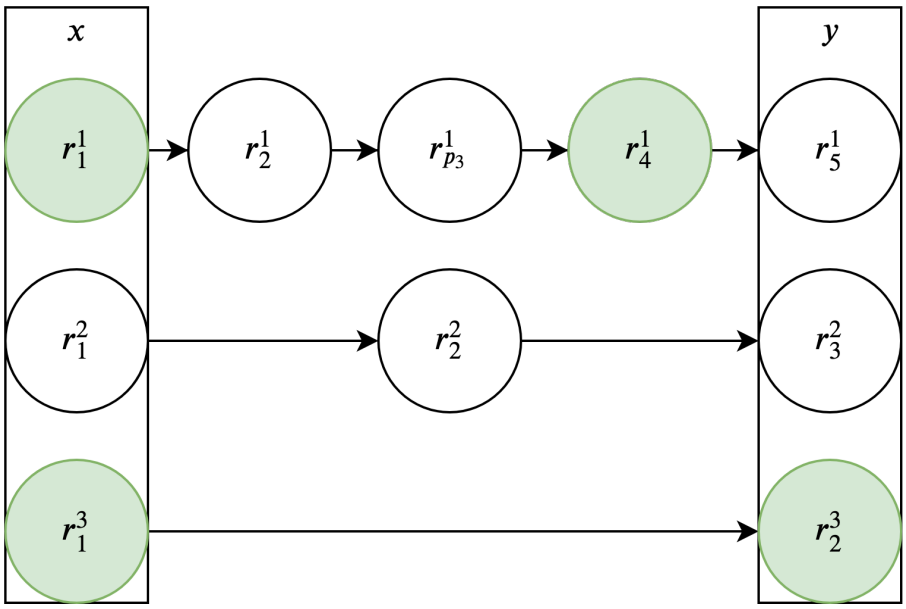


Figura 3.18: Um exemplo de marcação de conectividade para um conjunto $V_{\mathfrak{C}}$. As amostras em verde estão dentro da zona de conectividade.

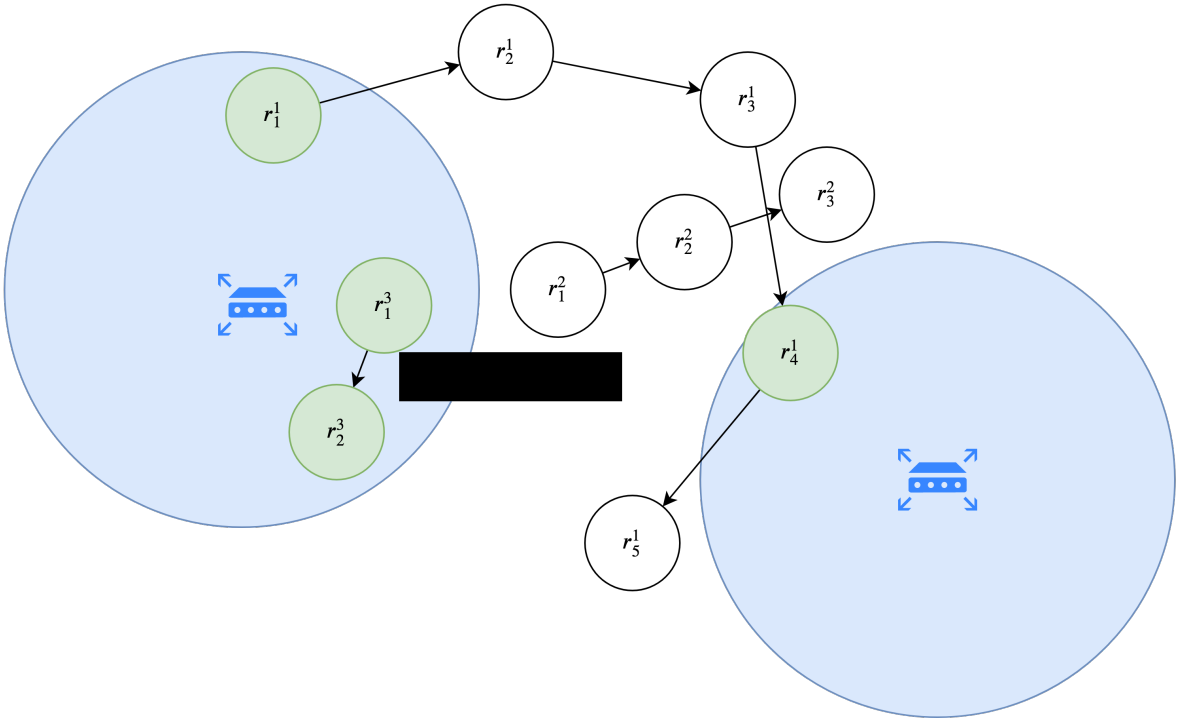


Figura 3.19: Exemplo de uma situação que representa o problema da Figura 3.18. O retângulo preto é um obstáculo.

Para definir se a configuração y é valida ou não, são utilizadas duas filas: a fila de conectividade $Q_{\mathfrak{C}}$ e a fila de incerteza $Q_{\mathfrak{U}}$. Para cada cara robô $r_i \in \mathbf{R}_t$, caso a posição $r^i_{n_i}$ tenha sido marcada no passo anterior, ou seja, ela está dentro da zona de conectividade, ela é inserida na fila $Q_{\mathfrak{C}}$. Caso contrário, $r^i_{n_i}$ é colocada dentro da fila $Q_{\mathfrak{U}}$, e para cada outro robô $r_j \in \mathbf{R}_t$ com $i \neq j$ é verificado se já existe uma aresta $e_{\text{vis}} \in E_{\text{vis}}$ ou $e_{\cancel{\text{vis}}} \in E_{\cancel{\text{vis}}}$, que represente a visibilidade ou não visibilidade entre as posições $r^i_{n_i}$ e $r^j_{n_j}$. Casos não exista nenhuma das duas, é executada uma análise de visibilidade entre $r^i_{n_i}$ e $r^j_{n_j}$, um vetor

$$\boldsymbol{e}_{\text{vis}} = r^j_{n_j} - r^i_{n_i} \tag{3.45}$$

é calculado e

$$n_{\text{vis}} = \left\lceil \frac{\|\boldsymbol{e}_{\text{vis}}\|}{\delta_{obs}} \right\rceil \tag{3.46}$$

amostras são retiradas desse vetor seguindo a fórmula

$$p_k = r^i_{n_i} + \frac{\boldsymbol{e}_{\text{vis}}}{\|\boldsymbol{e}_{\text{vis}}\|} \frac{k}{n_{\text{vis}}}, \tag{3.47}$$

em que p_k é a k-ésima amostra. Caso nenhuma dessas n_{vis} amostras p_k tenha um valor $\mathcal{O}(p_k) = 1$, então a aresta $\{r^i_{n_i}, r^j_{n_j}\}$ é adicionada a E_{vis}, caso contrário, ela é adicionada a $E_{\cancel{\text{vis}}}$. Se uma determinada posição $r^i_{n_i}$ não possua visibilidade para nenhuma outra posição, então quer dizer que esse ponto não terá conexão e portanto existe uma colisão. Caso contrário, os grafos de visibilidade e não visibilidade para a configuração y estão montados e pode-se passar para a análise das filas $Q_{\mathfrak{C}}$ e $Q_{\mathfrak{U}}$. Se $|Q_{\mathfrak{C}}|$ for igual a $|\mathbf{R}_t|$, então todos os robôs estão conectados e a configuração y é valida. Caso contrário, para cada posição $r^i_{n_i} \in Q_{\mathfrak{C}}$, analisa-se cada posição $r^j_{n_j} \in Q_{\mathfrak{U}}$, caso exista um vértice no grafo de visibilidade que conecte essas duas posições, $r^j_{n_j}$ é removido de $Q_{\mathfrak{U}}$ e colocado no final de $Q_{\mathfrak{C}}$. Quando todos os elementos de $Q_{\mathfrak{C}}$, incluindo os novos que

estão sendo adicionados, forem processados, analisa-se a fila $Q_{\mathfrak{U}}$. Caso ela esteja vazia, todos os pontos foram conectados e a configuração y é valida, caso contrário, existem pontos sem conexão e portanto y é invalida. Se a configuração y é inválida, então a aresta $e = \{x, y\}$ também é invalida e contém uma colisão.

Se a aresta $e = \{x, y\}$ não pertence a nenhum dos casos especiais acima, é necessário realizar a análise mais completa e complexa de conectividade. A ideia de como analisar essa conectividade deriva da análise de conectividade de apenas uma configuração, só que ao invés de utilizar apenas as amostras de y, são utilizadas todas as amostras de $V_{\mathfrak{C}}$. Novamente utiliza-se as filas de conectividade $Q_{\mathfrak{C}}$ e de incerteza $Q_{\mathfrak{U}}$. Só que agora, para cada robô $r_i \in \mathbf{R}_t$ deve-se analisar todas as posições r_j^i com $j \in \{1, 2, \ldots, n_i\}$, ao invés de apenas a última posição. Caso a posição r_j^i esteja na zona de conectividade, ela é colocada dentro da fila de conectividade $Q_{\mathfrak{C}}$. Caso contrário, ela é colocada na fila de incerteza $Q_{\mathfrak{U}}$ e para cada outra k-ésima posição de outro robô $r_k^l \in \mathbf{R}_t$ com $i \neq l$, faz se a verificação se já existe alguma aresta de visibilidade ou de não visibilidade, computando uma caso nenhuma das duas exista, de acordo com as Equações 3.45 a 3.47, apenas trocando $r_{n_j}^j$ por r_k^l e $r_{n_i}^i$ por r_j^i. Novamente, caso alguma posição r_j^i não esteja na zona de conectividade e nem tenha visibilidade para nenhuma outra posição, essa amostra é desconexa e a análise termina, pois não é possível executar o movimento sem perda de conexão. Caso todas as posições possuam ao menos uma conexão no grafo de visibilidade, se a fila $Q_{\mathfrak{U}}$ estiver vazia, essa aresta $e = \{x, y\}$ não possui nenhuma colisão e pode ser realizada sem perda de conexão e sem colisão física com obstáculos. Se Q_u não estiver vazia, exatamente a mesma análise do caso individual é feita, conectando os pontos incertos que têm visibilidade a pontos de conexão da fila $Q_{\mathfrak{C}}$. Após esse passo, caso Q_u esteja vazio $e = \{x, y\}$ não possui colisão, e caso contrário ela possui colisão e não pode ser utilizada.

Para um melhor entendimento, será feita uma rápida análise do exemplo da Figura 3.19. O primeiro passo é obter as amostras $V_{\mathfrak{C}}$ representadas na Figura 3.18 e marcar aquelas que estejam dentro da zona de conectividade, bolas verdes. Como nenhuma delas colide com um obstáculo, e o robô r_3 está completamente dentro de uma zona de conectividade, mas existem pontos fora dessa zona é necessário fazer a análise do grafo de visibilidade para a amostra y. Iniciando pela posição r_5^1, ela não se encontra na zona de conectividade, e portanto, é adicionada a $Q_{\mathfrak{U}}$. Essa executa a rotina de verificação de visibilidade para ambas as outras amostras e descobre que existe visibilidade entre r_5^1 e as outras, portanto, E_{vis} se torna $E_{\text{vis}} = \{\{r_5^1, r_3^2\}, \{r_5^1, r_2^3\}\}$. O mesmo ocorre à amostra r_3^2, ela é adicionada a $Q_{\mathfrak{U}}$, porém ela não precisa executar a rotina de verificação entre r_3^2 e r_5^1, pois essa aresta já existe em E_{vis}, ela verifica apenas a aresta para r_2^3 é descobre que ela pertence a $E_{\not{\text{vis}}}$. Já a amostra r_2^3 se encontra dentro da zona de conectividade e por causa disso é apenas inserida em $Q_{\mathfrak{C}}$.

Sendo assim, ao final da primeira parte da verificação se y é valida tem-se

$$Q_{\mathfrak{C}} = \{r_2^3\}, \tag{3.48}$$

$$Q_{\mathfrak{U}} = \{r_5^1, r_3^2\}, \tag{3.49}$$

$$E_{\text{vis}} = \{\{r_5^1, r_3^2\}, \{r_5^1, r_2^3\}\} \text{ e} \tag{3.50}$$

$$E_{\not{\text{vis}}} = \{\{r_3^2, r_2^3\}\}. \tag{3.51}$$

Para a segunda parte analisa-se as posições em $Q_{\mathfrak{C}}$ e tenta-se ligá-las por meio de uma aresta de visibilidade. A primeira posição de $Q_{\mathfrak{C}}$ é r_2^3, a única aresta de visibilidade que contêm essa posição com alguma de $Q_{\mathfrak{U}}$ é a aresta $\{r_5^1, r_2^3\}$. Sendo assim, r_5^1 é removido de $Q_{\mathfrak{U}}$ e colocado em $Q_{\mathfrak{C}}$. Agora a segunda posição de $Q_{\mathfrak{C}}$ é r_5^1 e a única aresta de visibilidade que contêm essa posição com alguma de $Q_{\mathfrak{U}}$ é a aresta $\{r_5^1, r_3^2\}$, portanto r_3^2 é removido de $Q_{\mathfrak{U}}$ e adicionado a $Q_{\mathfrak{C}}$. Como $Q_{\mathfrak{U}}$ está vazia, a configuração y é válida. Portanto, agora, basta fazer a última verificação. Para isso a Tabela 3.1, contendo um resumo de qual arestas entre posições de robôs

Tabela 3.1: Representação em forma de tabela das aresta que possuem ou não visibilidade para a Figura 3.19.

$\times$	r_1^1	r_2^1	r_3^1	r_4^1	r_5^1	r_1^2	r_2^2	r_3^2	r_1^3	r_2^3
r_1^1	$\times$	$\times$	$\times$	$\times$	$\times$	S	S	S	S	S
r_2^1	$\times$	$\times$	$\times$	$\times$	$\times$	S	S	S	S	S
r_3^1	$\times$	$\times$	$\times$	$\times$	$\times$	S	S	S	S	N
r_4^1	$\times$	$\times$	$\times$	$\times$	$\times$	S	S	S	S	N
r_5^1	$\times$	$\times$	$\times$	$\times$	$\times$	N	S	S	N	S
r_1^2	$\times$	$\times$	$\times$	$\times$	$\times$	$\times$	$\times$	$\times$	S	N
r_2^2	$\times$	$\times$	$\times$	$\times$	$\times$	$\times$	$\times$	$\times$	S	N
r_3^2	$\times$	$\times$	$\times$	$\times$	$\times$	$\times$	$\times$	$\times$	S	N
r_1^3	$\times$	$\times$	$\times$	$\times$	$\times$	$\times$	$\times$	$\times$	$\times$	$\times$
r_1^3	$\times$	$\times$	$\times$	$\times$	$\times$	$\times$	$\times$	$\times$	$\times$	$\times$

diferentes possuem ou não visibilidade, será utilizada para facilitar essa verificação.

Após a primeira etapa do algoritmo as filas e arestas do grafo de visibilidade estarão dessa maneira

$$Q_{\mathfrak{C}} = \{r_1^1, r_4^1, r_1^3, r_2^3\}, \tag{3.52}$$

$$Q_{\mathfrak{U}} = \{r_2^1, r_3^1, r_5^1, r_1^2, r_2^2, r_3^2\}, \tag{3.53}$$

$$\begin{aligned} E_{\text{vis}} = \{&\{r_2^1, r_1^2\}, \{r_2^1, r_2^2\}, \{r_2^1, r_3^2\}, \{r_2^1, r_1^3\}, \{r_2^1, r_2^3\}, \\ &\{r_3^1, r_1^2\}, \{r_3^1, r_2^2\}, \{r_3^1, r_3^2\}, \{r_3^1, r_1^3\}, \\ &\{r_5^1, r_2^2\}, \{r_5^1, r_3^2\}, \{r_5^1, r_2^3\}, \\ &\{r_1^2, r_1^3\}, \{r_1^2, r_1^1\}, \\ &\{r_2^2, r_1^3\}, \{r_2^2, r_1^1\}, \\ &\{r_3^2, r_1^3\}, \{r_3^2, r_1^1\}\}. \end{aligned} \tag{3.54}$$

Após processar o primeiro elemento de $Q_{\mathfrak{C}}$, as filas mudam para

$$Q_{\mathfrak{C}} = \{r_1^1, r_4^1, r_1^3, r_2^3, r_1^2, r_2^2, r_3^2\} \text{ e} \tag{3.55}$$

$$Q_{\mathfrak{U}} = \{r_2^1, r_3^1, r_5^1\}. \tag{3.56}$$

A próxima mudança acontece na posição r_1^3,

$$Q_{\mathfrak{C}} = \{r_1^1, r_4^1, r_1^3, r_2^3, r_1^2, r_2^2, r_3^2, r_2^1, r_3^1, \} \text{ e} \tag{3.57}$$

$$Q_{\mathfrak{U}} = \{r_5^1\}. \tag{3.58}$$

E finalmente em r_2^3

$$Q_{\mathfrak{C}} = \{r_1^1, r_4^1, r_1^3, r_2^3, r_1^2, r_2^2, r_3^2, r_2^1, r_3^1, r_5^1\} \text{ e} \tag{3.59}$$

$$Q_{\mathfrak{U}} = \{\}. \tag{3.60}$$

Como $Q_{\mathfrak{U}}$ está vazio, não é necessário fazer nenhuma iteração e se pode considerar que a transição das amostras x para a amostra y da Figura 3.18 pode ocorrer sem nenhuma colisão ou perda de comunicação. Um exemplo disso é que o robô r_1 se desloca de r_1^1 até r_4^1, enquanto o robô r_2 se desloca de r_1^2 até r_3^2. Ambos estão mantendo

sua conexão por meio do robô r_3. Quando ambos chegam a esses destinos, o robô r_3 se desloca de r_1^3 para r_2^3. A conexão de r_2 durante esse tempo vêm de r_1 que está em uma zona de conectividade. Quando r_3 chega a seu destino, r_1 se desloca de r_4^1 para r_5^1, com o robô r_3 garantindo a sua conexão. É importante frisar que para que esse método funcione δ_{obs} deve ser suficientemente pequeno.

Utilizando o espaço de configuração definido nesta seção, juntamente com a função de distância $\rho_{2|\mathbf{R}_t|}(a, b)$, a função de amostragem $\alpha_{2|\mathbf{R}_t|}(i)$ e a metodologia de detecção de colisão apresentados, é possível utilizar os algoritmos de busca em árvore apresentados na Seção 2.4.5 para gerar um plano de execução de uma tarefa t por um conjunto de robôs $\mathbf{R}_t$. Completando o módulo de coordenação de missão.

4 RESULTADOS

4.1 INTRODUÇÃO

Este capítulo visa realizar uma série de testes para avaliar e validar o arcabouço de planejamento de movimentação e realização de tarefas com restrição de movimento proposto no Capítulo 3. A ideia inicial é que esses testes seriam realizados pelos robôs Pioneers se movimentando realmente pelo LARA. Infelizmente, isso não foi possível, devido a restrições de tempo para finalização de reparo nos robôs, implementação de um sistema de navegação confiável e desenvolvimento e implementação do módulo de execução do movimento.

Em função disso, os testes irão focar apenas no trabalho que foi desenvolvido e implementado: os módulos de planejamento de missão e de coordenação de missão. Inicialmente serão feitos alguns testes que mostrarão situações e qualidades específicas para a geração de tarefas-α. Depois disso, serão realizados alguns ensaios relacionados ao cálculo de custo de tarefas por meio de heurísticas com o objetivo de compará-las. Por fim, um cenário parecido com uma parte do primeiro andar do SG-11 será utilizado para modelar uma situação próxima do que se espera para o funcionamento completo do sistema.

Apesar do sistema receber situações simuladas, os componentes que serão testados já são as versões que fariam parte do sistema final, uma vez que esses componentes fazem parte do servidor remoto que os robôs acessariam. Os arquivos de entrada e de saída já são os que seriam trocados entre os robôs e o servidor remoto. Esse sistema foi desenvolvido utilizando o C++17 e está disponível em um repositório online[1]. As imagens utilizadas para representar as situações foram geradas pela ferramenta *gnuplot*[2].

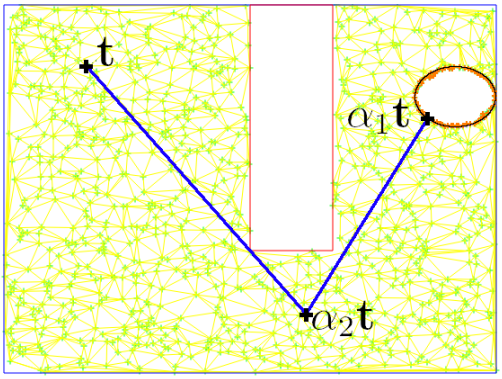
4.2 TAREFAS-α

Para realizar uma análise do comportamento e das capacidades do algoritmo proposto na Seção 3.3, cinco cenários diferentes foram testados. Os cenários a e b, Figura 4.1, são os mais simples entre eles, com apenas um obstáculo retangular e uma região de conectividade. O cenário c, Figura 4.2, possui uma passagem bem estreita para o ambiente. O cenário d, Figura 4.3a, possui duas regiões de conectividade e o cenário e, Figura 4.3b, é o mais complexo, sendo composto por duas regiões de conectividade e uma tarefa bem no meio de uma espiral.

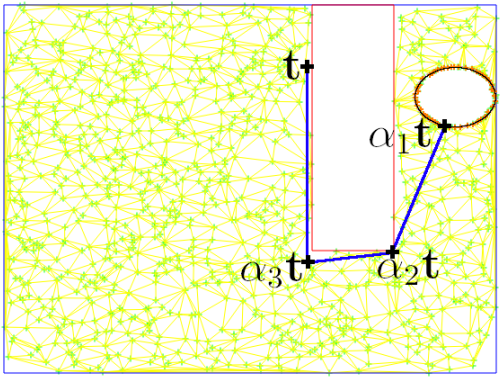
Os pontos verdes, das Figuras 4.1 até a Figura 4.3, são as amostras V utilizadas para a construção do mapa de busca $\mathcal{M}(V, E, \mathcal{D})$. As linhas amarelas são as arestas E desse mapa, calculadas pela triangulação de Delaunay. Os retângulos vermelhos são a representação da área obstruída $\mathcal{W}_O$. Os círculos pretos são as regiões nas quais a função de conectividade $\mathfrak{C} : \mathcal{W} \rightarrow [0, 1]$ possui um valor maior que o limiar mínimo $\gamma_{\mathfrak{C}}$. As cruzes pretas são as tarefas ou tarefas-α que devem ser realizadas pelos robôs. As $k \in \mathbb{N}$ tarefas-α que uma determinada tarefa t pode ter são representadas como $\alpha_i t$, em que $i = 1, \ldots, k$. A primeira tarefa-α, $\alpha_1 t$, é sempre a tarefa que fica na borda da região de conectividade. Tarefas-α e sua tarefa primária são conectadas por meio de uma linha azul. O retângulo azul ao redor representa a área de busca $\mathcal{W}_B$.

[1] https://github.com/rodrigowerberich/MultiTaskAllocation

[2] *http://www.gnuplot.info/*



(a) Cenário a: Um simples obstáculo retangular.



(b) Cenário b: O obstáculo retangular está muito mais próximo de ambas a tarefa e a região de conectividade.

Figura 4.1: Os primeiros dois cenários de teste para as tarefas-α. Os pontos verdes representam os pontos de amostragem utilizados para construir o espaço de busca. Os segmentos de reta amarelos são a triangulação de Delaunay que conecta esses pontos. O retângulo vermelho representa um obstáculo. O círculo verde é a zona de conectividade real, enquanto que os pontos laranjas representam a estimação dessa zona de conectividade. As cruzes pretas são as tarefas ou tarefas-α que foram calculadas. As linhas azuis conectam as tarefas-α em sequência até a tarefa principal, mostrando a visibilidade entre as tarefas.

O primeiro aspecto interessante a se notar é que a região de conectividade não é conhecida a priori pelo algoritmo. Mesmo assim, o algoritmo conseguiu estimar de maneira suficientemente acurada a borda dessa região por meio das formas-α. Em cada uma das situações, os pontos α-extremos que formam a borda da estimação da região de conectividade podem ser visualizados ao redor dos círculos de conectividade. Essa representação pelos círculos pretos serve apenas como referência para o leitor poder entender melhor a situação, o algoritmo possui apenas a função $\mathfrak{C} : \mathcal{W} \to [0, 1]$ para realizar sua amostragem.

Os cenários mais simples, a e b, demonstram a capacidade do algoritmo de calcular o menor número de

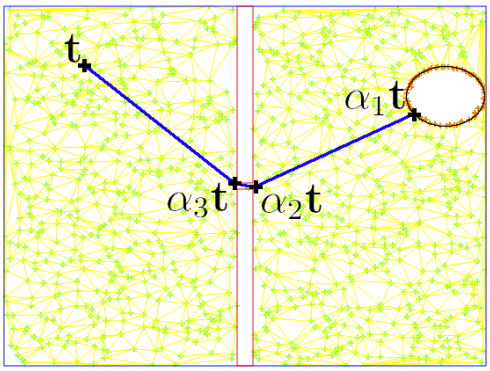
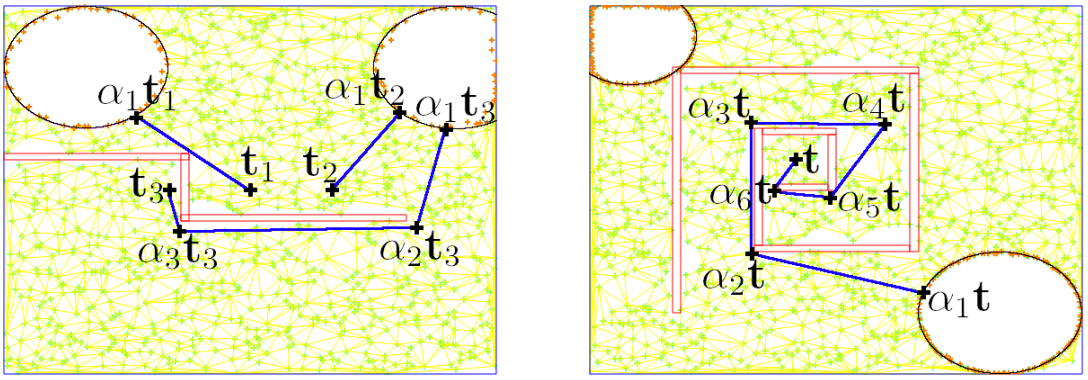


Figura 4.2: Cenário c: Uma passagem muita estreita entre a região onde se encontra a tarefa e a região de conectividade.



(a) Cenário d: Quando existe mais de uma região de conectividade, cada tarefa será conectada àquela que está mais próxima conside- rando o desvio de obstáculos.

(b) Cenário e: Ambiente complexo, composto por uma espiral e mais de uma região de conectividade.

Figura 4.3: Os últimos dois cenários de teste para as tarefas-α.

tarefas auxiliares, e por consequência o menor número de robôs necessários para realizar uma determinada tarefa considerando as restrições geométricas do ambiente e de sua área de busca $\mathcal{W}_B$. No cenário a, Figura 4.1a, o ambiente possui espaço suficiente para que a tarefa t e a região de conectividade sejam conectados por apenas um ponto. Desta forma, apenas duas tarefas-α são necessárias para realizar a tarefa t. Já no cenário b, Figura 4.1b, esse ponto único de conexão ficaria fora da área de busca $\mathcal{W}_B$, e portanto é considerado inválido. Em vista disso, esse cenário necessita de uma tarefa-α adicional, em relação ao cenário a.

O único caminho existente entre a tarefa t e a região de conectividade no cenário c, Figura 4.2, é uma passagem bem estreita. Essa passagem estreita representa $1,7\%$ da altura total da área de busca $\mathcal{W}_B$. Uma brecha desse tamanho não possui nenhum tipo de impacto no algoritmo, que conseguiu em todas as iterações rodadas encontrar sempre as três tarefas-α necessárias para manter uma conexão. Entretanto, quando essa brecha é reduzida a apenas $0,7\%$ da altura total, o algoritmo começou a apresentar instabilidades e em apenas 50% das iterações executadas foi possível encontrar uma resposta. Nos outros 50% o algoritmo retornava que era impossível alcançar alguma região de conectividade a partir da tarefa t. Isso significa que em um ambiente cuja altura seja 100 metros, o algoritmo teria problemas com buracos de 70 cm ou menos. É muito importante frisar que esses experimentos foram realizados com apenas 1000 amostras do ambiente, caso mais amostras sejam utilizadas, esses buracos podem ser reduzidos, porém existe uma penalidade de um maior custo computacional de processamento.

No cenário d, Figura 4.3a, foi exemplificada a capacidade do algoritmo de lidar com mais de uma região de conectividade. Para o algoritmo, na verdade, não faz a menor diferença se existem uma, duas ou n regiões de conectividade. Ele tenta se conectar com o ponto α-extremo mais próximo da tarefa t por meio de um caminho livre de colisões. Visualmente parece que a tarefa t se conecta a zona de conectividade mais próxima, respeitando as restrições geométricas impostas pelos obstáculos físicos. Por exemplo, na Figura 4.3a a tarefa t_1 se encontra mais próxima da região de conectividade da esquerda, enquanto a tarefa t_2 está mais próxima da região da direita, e conforme esperado, elas são conectadas pelas suas tarefas-α às suas respectivas regiões. Entretanto, por mais que t_3 pareça estar mais próxima da região da esquerda, ela é conectada a região de conectividade da direita, pois ao se contornar o obstáculo entre t_3 e a primeira, t_3 acaba ficando mais próxima da segunda.

O cenário e, Figura 4.3b, possui a maior complexidade geométrica entre todos eles. Ele possui uma espiral com uma tarefa t em seu centro com duas regiões de conectividade. O algoritmo não apresentou dificuldades em calcular as tarefas-α associadas a t. A princípio o algoritmo foi acompanhando a curva da espiral, mas conseguiu gerar um ponto mais afastado $\alpha_4 t$ que permitiu que a tarefa t fosse realizada com uma tarefa-α a menos do que seria necessário caso ele apenas continuasse seguindo a espiral.

Com as tarefas-α é possível determinar o número de robôs necessários para realizar cada um dos cenários apresentados. A Tabela 4.1 apresenta essa quantidade para cada um dos cenários. Para o cenário d, são apresentados três números. Cada um desses números representa uma das diferentes tarefas que compõem esse cenário. Esses números foram obtidos da quantidade de tarefas-α de t, somado a um $\mathcal{N}_\alpha(t) + 1$, conforme apresentado na Seção 3.3.

Tabela 4.1: Número de robôs necessários para realizar as tarefas dos cenários apresentados nas Figuras 4.1 a 4.3.

Cenário	Número de robôs
a	3
b	4
c	4
d	2,2,4
e	7

Analisando, principalmente, os cenários d e e é possível reforçar o porquê o modelo de visibilidade foi escolhido. O modelo radial no caso d determinaria para t_3 algo entre um ou duas tarefas-α, mas não indicaria se é possível contornar o obstáculo com esse número de robôs. O mesmo aconteceria com o cenário e, indo em diagonal para alguma das zonas de conectividade, seria possível conectar a tarefa t por meio de duas ou três tarefas-α, mas novamente, não existe garantia de que seria possível que esses robôs chegassem no ponto que eles precisam chegar dentro da espiral para garantir a conexão de t.

O modelo de visibilidade consegue garantir que as tarefas-α obtidas são realizáveis sem perda de comunicação, pois elas podem ser utilizadas como um plano simples de como executar a tarefa t sem que nenhum robô se desconecte. Considerando que todos os robôs se encontram dentro da zona de conectividade da primeira tarefa-α, cada robô deve percorrer as tarefas-α em ordem. Por exemplo, no cenário a três robôs x, y e z começariam dentro da zona de conectividade. Os robôs x, y e z se deslocariam até estarem todos em $\alpha_1 t$, quando todos estiverem lá, z ficaria para trás enquanto x e y se deslocam até $\alpha_2 t$. Ao alcançá-la, x continuaria sozinho até t para executar a tarefa enquanto a conexão é mantida.

4.3 COMPARANDO HEURÍSTICAS

O segundo tipo de teste realizado foi o de comparação entre as diferente heurísticas apresentadas na Seção 3.4. Para compará-las, foi criado um cenário hipotético, representado na Figura 4.4. Esse cenário é composto por três robôs $\mathbf{R} = \{r_1, r_2, r_3\}$, representados pelos triângulos roxos, e três missões $\mathbf{M} = \{m_1, m_2, m_3\}$. As três missões possuem a mesma prioridade e o mesmo prazo de finalização. Cada um dos robôs é de um tipo diferente $\mathcal{T}_R = \{\tau_{R1}, \tau_{R2}, \tau_{R3}\}$ e existem três tipos de tarefas $\mathcal{T}_\mathbf{T} = \{\tau_{T1}, \tau_{T2}, \tau_{T3}\}$. A função de esforço é

dada de acordo com a matriz

$$e = \begin{array}{c} \begin{matrix} \tau_{R1} & \tau_{R2} & \tau_{R3} \end{matrix} \\ \begin{bmatrix} 8 & 5 & 6 \\ 6 & 8 & 10 \\ 7 & 10 & 5 \end{bmatrix} \end{array} \begin{matrix} \tau_{T1} \\ \tau_{T2} \\ \tau_{T3} \end{matrix} , \tag{4.1}$$

em que as colunas correspondem ao tipo do robô e as linhas ao tipo da tarefa.

A missão m_1, representada pela tarefa em vinho, possui apenas uma tarefa t_1. Essa tarefa é do tipo $t_\tau^1 = \tau_{T1}$, e se encontra depois de uma parede , por causa disso, necessita de duas tarefas auxiliares $\alpha_1 t_1$ e $\alpha_2 t_2$. A missão m_2, representada pelas tarefas em verde, possui duas tarefas t_2, do tipo $t_\tau^2 = \tau_{T2}$, e t_3, do tipo $t_\tau^3 = \tau_{T2}$, cada uma delas dispõe de suas respectivas tarefas-α $\alpha_1 t_2$ e $\alpha_1 t_3$. Ambas essas tarefas auxiliares se encontram exatamente no mesmo local. A terceira missão m_3, caracterizada pelas tarefas de cor azul, contém quatro tarefas: t_4, do tipo $t_\tau^4 = \tau_{T2}$; t_5, do tipo $t_\tau^5 = \tau_{T2}$; t_6, do tipo $t_\tau^6 = \tau_{T2}$; e t_7, do tipo $t_\tau^7 = \tau_{T3}$. As quatro dentro da zona de conectividade. Essas tarefas foram estabelecidas de forma a criar casos diferentes para as funções heurísticas examinarem. A primeira m_1, por possuir uma tarefa atrás de um obstáculo e mais distante da zona de conectividade. A segunda m_2 por conter duas tarefas com linha de visada, porém fora da zona de conectividade. E a terceira m_3, por conter um número maior de tarefas.

A Tabela 4.2 possui um resumo da execução do algoritmo de decisão para todas as cinco diferentes heurísticas apresentadas na Seção 3.4. O valor c_T corresponde ao custo total, e é dado como

$$c_T(m_j) = (\sum_{i=1}^{N} \left(h\left(t_i\right) + \mathbf{e}\left(t_\tau^i, r_\tau^i\right)\right), \tag{4.2}$$

o que corresponde a parte que subtrai a folga na Equação 3.20. O valor utilizado para a constante k, tanto para h_{2a}, como para h_{2b}, foi de $k = 2$.

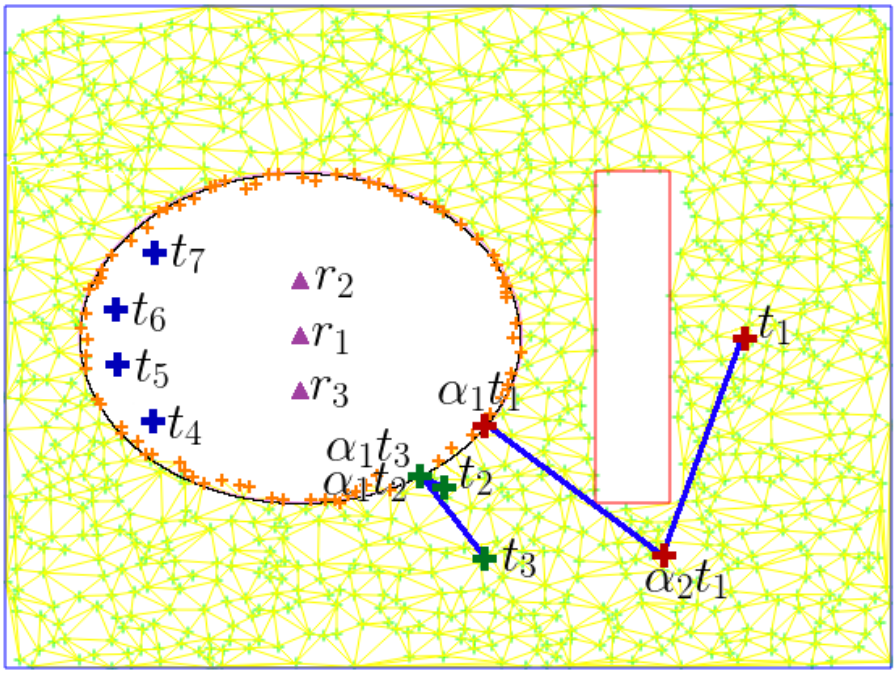


Figura 4.4: Um cenário com três missões (tarefas em vinho, verde e azul) e três robôs.

Tabela 4.2: Valores dos custos totais, prazos e folgas para cada uma das missões utilizando as diferentes heurísticas.

—	m_1			m_2			m_3			—
Heurística	$c_T(m_1)$	$m_\mathfrak{d}^1$	$\mathfrak{f}(m_1)$	$c_T(m_2)$	$m_\mathfrak{d}^2$	$\mathfrak{f}(m_2)$	$c_T(m_3)$	$m_\mathfrak{d}^3$	$\mathfrak{f}(m_3)$	Escolhido
h_1	$423,21$	60	$-363,21$	$15,13$	60	$44,82$	$23,00$	60	$37,00$	m_1
h_{2a}	$26,20$	60	$33,80$	$23,50$	60	$36,50$	$30,74$	60	$29,26$	m_3
h_{2b}	$34,09$	60	$25,91$	$23,50$	60	$36,50$	$30,74$	60	$29,26$	m_1
h_{2c}	$33,11$	60	$26,89$	$28,12$	60	$30,74$	$30,74$	60	$29,26$	m_1
h_{2d}	$50,06$	60	$9,94$	$28,12$	60	$30,74$	$30,74$	60	$29,26$	m_1

Analisando a Tabela 4.2, percebe-se que com exceção da heurística h_{2a}, todas obtiveram o mesmo resultado final, escolhendo a missão m_1 como a missão que deve ser executada. A missão m_2 sempre ficou em último lugar para todas as heurísticas, enquanto que a missão ficou m_3 ficou em segundo lugar para todas, menos no caso em que ela ganhou de m_1.

O primeiro fato interessante a se notar é o motivo pelo qual a missão m_2 sempre apresenta uma folga maior do que m_3. Isso ocorre, pois a missão m_3 possui muitas tarefas, e como o esforço para realizar as tarefas está na mesma ordem de grandeza dos deslocamentos necessários para realizar tanto as tarefas de m_1 como as de m_3, os esforços acabaram sendo um fator determinante entre essas duas missões. Como curiosidade, foi diminuído o esforço necessário para realização de cada uma das tarefas e nesse novo cenário, m_2 sempre acabava com uma folga menor do que m_3.

O segundo fato interessante é que a heurística h_1 acaba por superestimar o custo de tarefas que estejam muito fora da zona de conectividade, mas acaba por subestimar tarefas que estejam apenas um pouco fora da zona de conectividade. Coincidentemente, ela acabou gerando resultados finais iguais aos das outras heurísticas.

Outro ponto a se notar é que todas as h_2 se comportaram exatamente iguais para a missão m_3. Esse era o esperado, dado que todas as tarefas que pertencem a essa missão se encontram dentro da zona de conectividade e assim todas as essas funções heurísticas recaem para a mesma função base de soma das distância euclidianas entre elas.

Já para a missão m_2, as heurísticas h_{2a} e h_{2b} apresentaram o mesmo resultado, e as heurísticas h_{2c} e h_{2d} também. Isso ocorre, pois para essa missão h_{2a} e h_{2b} recaem para a função $\rho_{\pi k}$ enquanto que as heurísticas h_{2c} e h_{2d} recaem para ρ/Ψ. A ausência de obstáculos faz com que apenas a parcela relativa a conectividade influencie o resultado final.

Para a missão m_1, todas as heurísticas forneceram valores diferentes. Isso ocorre pois essa tarefa está tanto fora da região de conectividade, como possui obstáculos em seu caminho. Sendo assim, a função heurística não é simplificada e, portanto, fornece valores mais representativos da situação real.

De forma a obter uma comparação de custo computacional, todas essas funções foram executadas 100 vezes cada, e foi extraído o tempo médio de computação dessas amostras. O resultado pode ser visualizado na Tabela 4.3 que mostra o tempo em microssegundos. Todas as execuções foram realizadas na mesma máquina e para intuito de análise, somente a ordem de grandeza dos resultados sera considerada.

Como esperado, a função heurística h_1 é a mais veloz, com resultados na ordem de $0,1\,ms$. As duas métricas, h_{2a} e h_{2c}, que tentam estimar apenas a conectividade possuem tempos na mesma ordem de grandeza,

Tabela 4.3: Tempos médios, em μs, de execução de 100 iterações, para o cálculo dos custos de cada uma das missões.

Heurística	m_1	m_2	m_3
h_1	$136,30$	$92,60$	$117,70$
h_{2a}	$5412,60$	$3893,80$	$1564,00$
h_{2b}	$350356,00$	$33244,40$	$10295,20$
h_{2c}	$3446,20$	$2423,60$	$1151,70$
h_{2d}	$313840,20$	$30024,30$	$8928,80$

por volta de $1ms$. Enquanto que as duas métricas que utilizam árvores aleatórias para calcular o caminho também estão na mesma ordem de grandeza, entre 10 e $100ms$.

Sendo assim, a escolha por métodos que possuam uma heurística mais detalhista, realmente possui um peso computacional bem maior. Entretanto, não está claro a vantagem entre os métodos já que todos eles retornaram exatamente o mesmo resultado. Principalmente considerando que se um $k > 2$ tivesse sido utilizado, até para h_{2a} a missão m_1 teria sido a escolhida para ser executada. A única coisa que se torna clara, é que de fato as heurísticas h_{2b} e h_{2d} apresentam um grau maior de representatividade, e por isso, é possível imaginar que existam cenário em que elas apresentariam resultados completamente diferentes das outras heurísticas. Como esse processamento irá ocorrer na nuvem, foi escolhido utilizar essas heurísticas mais pesadas de forma a manter uma representação mais acurada do que pode acontecer.

Uma última análise a se fazer é a de qual robô realiza qual tarefa. Nessa análise serão consideradas apenas as heurísticas h_2 em que a missão escolhida foi a m_1. Para a função h_{2b} o robô r_1 ficou responsável pela tarefa $\alpha_1 t_1$, o robô r_3 ficou responsável pela tarefa $\alpha_2 t_1$ e o robô r_2 ficou responsável pela tarefa t_1. Essas escolhas refletem a constante k escolhida, os robôs r_1 e r_3 tentam ficar o máximo possível dentro da região de conectividade. O robô r_2 foi escolhido para realizar t_1, pois $k = 2$ não penaliza tanto andar fora da região de conectividade, então ele provavelmente obteve uma rota por cima do obstáculo de menor tamanho obtendo um menor custo total.

Para a função h_{2c} a ordem foi: robô r_3, tarefa $\alpha_1 t_1$; robô r_1, tarefa $\alpha_2 t_1$; e robô r_2, tarefa t_1. Essa escolha faz sentido para essa heurística, pois como ela ignora obstáculos, a distância dos robôs para as tarefas é aproximadamente a mesma, por isso, a função de esforço acaba contando mais, e como r_2 possui o menor esforço para realizar essa tarefa, ele foi o escolhido. A tarefa r_3 foi escolhida para executar a tarefa $\alpha_1 t_1$, porque a porcentagem de caminho que ela faria por fora da região de conectividade para realizar a tarefa $\alpha_2 t_1$ é maior do que a porcentagem para o robô r_1.

Pra a função h_{2d} a ordem foi: robô r_2, tarefa $\alpha_1 t_1$; robô r_1, tarefa $\alpha_2 t_1$; e robô r_3, tarefa t_1. Essa escolha foi bem diferente das outras, com r_3 realizando a tarefa t_1, mesmo não sendo o robô que executaria o menor esforço para essa tarefa. Isso ocorreu, pois a heurística h_{2d} tenta manter os robôs o máximo possível dentro da zona de conectividade, mesmo que os robôs tenham que percorrer um caminho maior, e nesse caso o custo de se manter dentro da zona de conectividade, mas utilizar um robô que precisa de mais esforço, foi suficientemente vantajoso comparado ao de utilizar o robô que executaria menos esforço na tarefa, mas no cenário em geral os robôs andariam mais fora da zona de conectividade.

4.4 SIMULANDO O SG11

O último teste visa criar uma situação similar ao que se pretende ser o modo de operação padrão desses sistema. Para isso foi modelada uma parte do primeiro andar do SG-11, Figura 4.5, por meio da linguagem descritora em JSON apresentada na Seção 3.2. O Apêndice A apresenta maiores detalhes desse arquivo de descrição.

Nesse cenário, existem os três robôs Pioneers $\mathbf{R} = \{r_1, r_2, r_3\}$, de tipos τ_{R1}, τ_{R2} e τ_{R3}, respectivamente. Os tipos de tarefas que existem refletem as capacidades dos robôs. Todos os robôs são capazes de pegar uma papel e entregar ele em algum lugar, tarefas relacionadas a isso são do tipo τ_I. O robô r_1 possui um adaptador específico para a cafeteira e, por causa disso, é o único capaz de buscar e levar café para pessoas. Tarefas relacionadas a cafeteria são do tipo τ_C. Os robôs r_2 e r_3 dispõem de um braço que possui integração com o depósito, como o robô r_1 teve de abandonar o braço para instalar o adaptador da cafeteira, ele é o único que não consegue buscar coisas no depósito. Tarefas relacionadas ao depósito são do tipo τ_D. Todos os robôs possuem uma mesa capaz de transportar algo que seja depositado nelas, tarefas desse tipo são classificadas como transporte e possuem o tipo τ_T. Sendo assim, $\mathcal{T}_\mathbf{T} = \{\tau_I, \tau_C, \tau_T, \tau_D\}$.

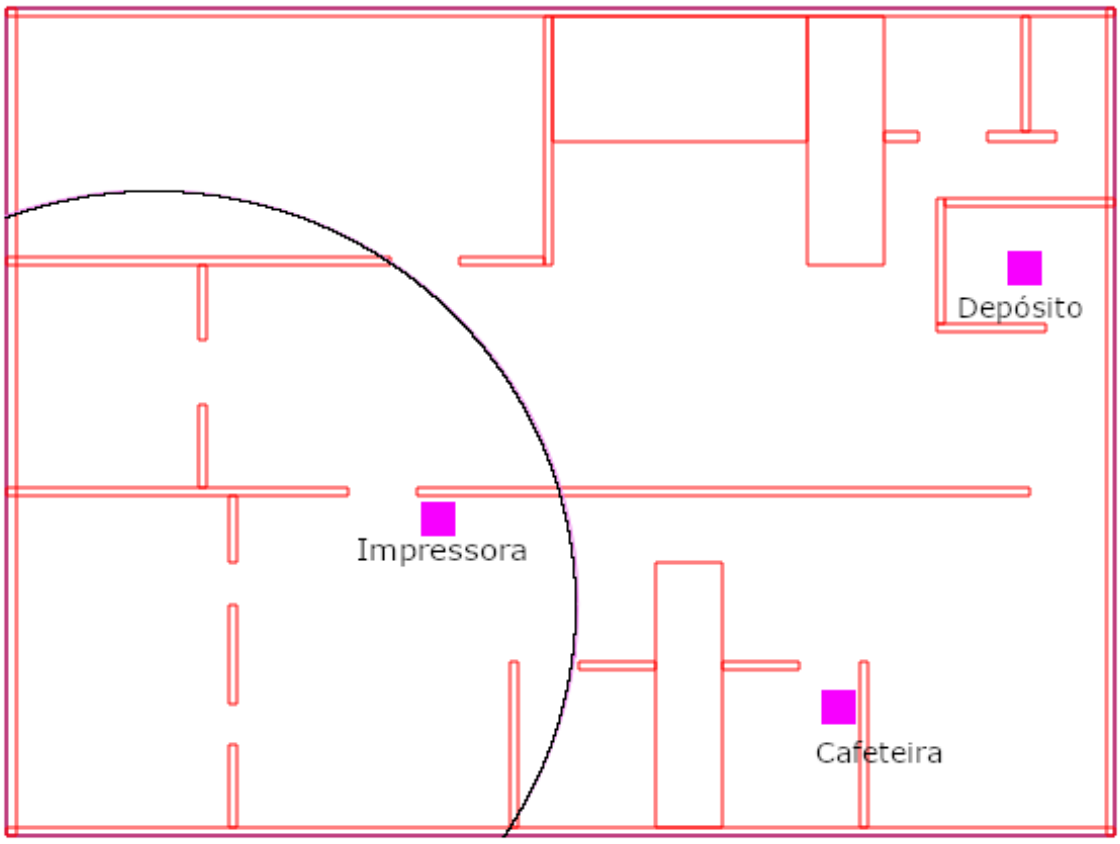


Figura 4.5: Um modelo de parte do primeiro andar do SG-11. A parte inferior da figura é o laboratório LARA.

A matriz

$$e = \begin{array}{c} \begin{matrix} \tau_{R1} & \tau_{R2} & \tau_{R3} \end{matrix} \\ \begin{bmatrix} 15 & 10 & 10 \\ 20 & \infty & \infty \\ \infty & 30 & 30 \\ 5 & 5 & 5 \end{bmatrix} \end{array} \begin{array}{c} \\ \tau_I \\ \tau_C \\ \tau_D \\ \tau_T \end{array} \tag{4.3}$$

representa a função de esforço $e : \mathcal{T}_{\mathbf{T}} \times \mathcal{T}_{\mathcal{R}} \to \mathbb{R}$, em que as linhas correspondem aos tipos de tarefas e as colunas aos tipos de robôs. Foi considerado que o robô r_1 possui uma maior dificuldade para pegar uma impressão devido ao adaptador para poder manipular a cafeteira. As tarefas que possuem um esforço infinito são tarefas impossíveis para aquele tipo de robô. Por causa disso, como os robôs r_2 e r_3 são incapazes de lidar com a cafeteira, seu esforço é infinito.

A fim de pegar uma impressão, um robô precisa se deslocar até a impressora que fica na posição $p_I = (-1,5; -1,2)$, representada por um retângulo rosa escrito "Impressora" na Figura 4.5. A cafeteira e o depósito estão marcados da mesma maneira, com a única diferença sendo que seus escritos estão apropriadamente "Cafeteira" e "Depósito". A cafeteira se encontra na posição $p_C = (3,3; -3,4)$ e o depósito na posição $p_D = (5,5; 1,5)$.

Com o objetivo de definir quais missões devem ser executadas, uma ordem de requisições, feita por pessoas fictícias, será simulada. Essas requisições são transformadas em missões pelo sistema, essas missões e as tarefas que fazem parte delas estão representadas na Figura 4.6.

Um determinado professor P_1 necessita de café em sua sala, que fica em $(0,5; -4)$. Isso pode ser traduzido para uma missão

$$m_1 = (\{t_2\}, 2, 300) \tag{4.4}$$

que é composta de duas tarefas sequenciais

$$t_1 = (\emptyset, p_C, \tau_C) \text{ e} \tag{4.5}$$

$$t_2 = (\{t_1\}, (0,5; -4), \tau_C)\,. \tag{4.6}$$

A tolerância para um robô pegar um café nesse sistema é de 5 min. Existem três níveis de prioridade nesse sistema. Tarefas essenciais requisitadas por um professor têm nível de prioridade 1, tarefas essenciais requisitadas por um aluno assim como tarefas não essenciais para professores têm prioridade 2 e, por fim, tarefas não essenciais para alunos têm nível de prioridade 3.

Um outro professor P_2 requisitou que uma peça do depósito seja entregue na sala de aula. Entregas no depósito têm um prazo limite de 10 min, sendo assim

$$m_2 = (\{t_4\}, 1, 600)\,, \tag{4.7}$$

$$t_3 = (\emptyset, p_D, \tau_D)\,, \tag{4.8}$$

$$t_4 = (\{t_3\}, (-1; 4), \tau_D)\,. \tag{4.9}$$

Um aluno A_1 requisitou que uma impressão fosse buscada para ele. O prazo para impressões é de 4 min, então

$$m_3 = (\{t_6\}, 2, 240)\,, \tag{4.10}$$

$$t_5 = (\emptyset, p_I, \tau_I)\,, \tag{4.11}$$

$$t_6 = (\{t_5\}, (-1,5; -4), \tau_I)\,. \tag{4.12}$$

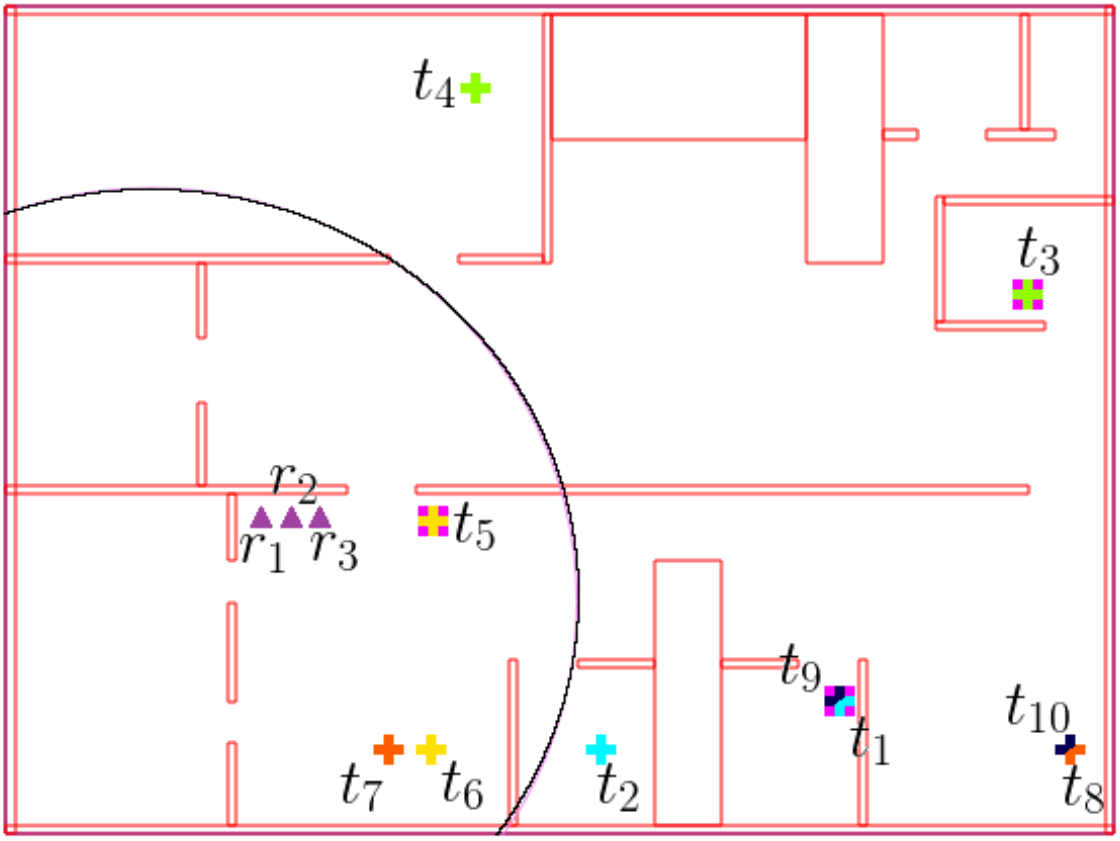


Figura 4.6: Posicionamento das tarefas e dos robôs. Tarefas que pertencem a mesma missão possuem a mesma cor.

Um outro aluno A_2 fez uma solicitação para que um item em sua posse fosse entregue para um outro aluno A_3. O prazo para entregas é de 10 minutos, por causa disso

$$m_4 = (\{t_8\}, 2, 600)\,, \tag{4.13}$$
$$t_7 = (\emptyset, (-2; -4), \tau_T)\,, \tag{4.14}$$
$$t_8 = (\{t_7\}, (6; -4), \tau_T)\,. \tag{4.15}$$

Por fim, o aluno A_3 requisitou café para ser entregue em sua mesa, então

$$m_5 = (\{t_{10}\}, 3, 300)\,, \tag{4.16}$$
$$t_9 = (\emptyset, p_C, \tau_C)\,, \tag{4.17}$$
$$t_{10} = (\{t_9\}, (6; -4), \tau_T)\,. \tag{4.18}$$

Cada uma dessas missões é representada por uma cor na Figura 4.6. Pontos que contenham mais de uma tarefa que sejam de missões diferentes são representados por mais de uma cor. A missão m_1 é representada

por um azul claro, m_2 por um verde claro, m_3 pela cor amarela, m_4 pela cor laranja e m_5 por um azul escuro, quase preto.

Considerando que todas essas missões foram recebidas ao mesmo tempo pelo servidor, o primeiro passo que ele irá executar é o planejamento de missão. Para isso, ele deve primeiramente gerar as tarefas-α associadas a cada uma das tarefas de todas essas missões. Após rodar o algoritmo, foi verificado que todas as tarefas eram executáveis com três robôs, por nenhuma delas exige mais do que duas tarefa-α, como pode se verificar na Figura 4.7. O tempo médio de cálculo foi por volta de 1 segundo, caso nenhum resultado fosse obtido após 60 segundos o algoritmo era encerrado e considerava-se que ele não conseguiu achar um resultado válido.

Na Figura 4.7, os pontos denominados $\alpha_m(t_i, t_j, \ldots, t_k)$ representam pontos que possuem múltiplas tarefas-α. As tarefas t_1, t_3, t_8, t_9 e t_{10} precisam de três robôs para executá-las. As tarefas t_2 e t_4 necessitam de dois robôs para poder ser executas, e as tarefa t_5, t_6 e t_7 podem ser realizadas por apenas um robô, pois se encontram dentro da zona de conectividade. Devido a estrutura do mapa de busca, tarefas que coexistem no mesmo ponto, geram tarefas auxiliares também em locais coexistentes, como por exemplo, as tarefas t_1 e t_9, ou as tarefas t_8 e t_9.

O próximo passo é calcular os custos e decidir qual será a primeira tarefa a ser executada. Para esse primeiro conjunto a decisão é simples, já que existe apenas uma tarefa com prioridade 1, então ela deve ser executada. Sendo assim, o planejador de missão escolhe a missão m_2 para ser passada para o módulo de coordenação de missão, para que ele possa gerar um caminho livre de colisões e de quedas de conexão até as tarefas.

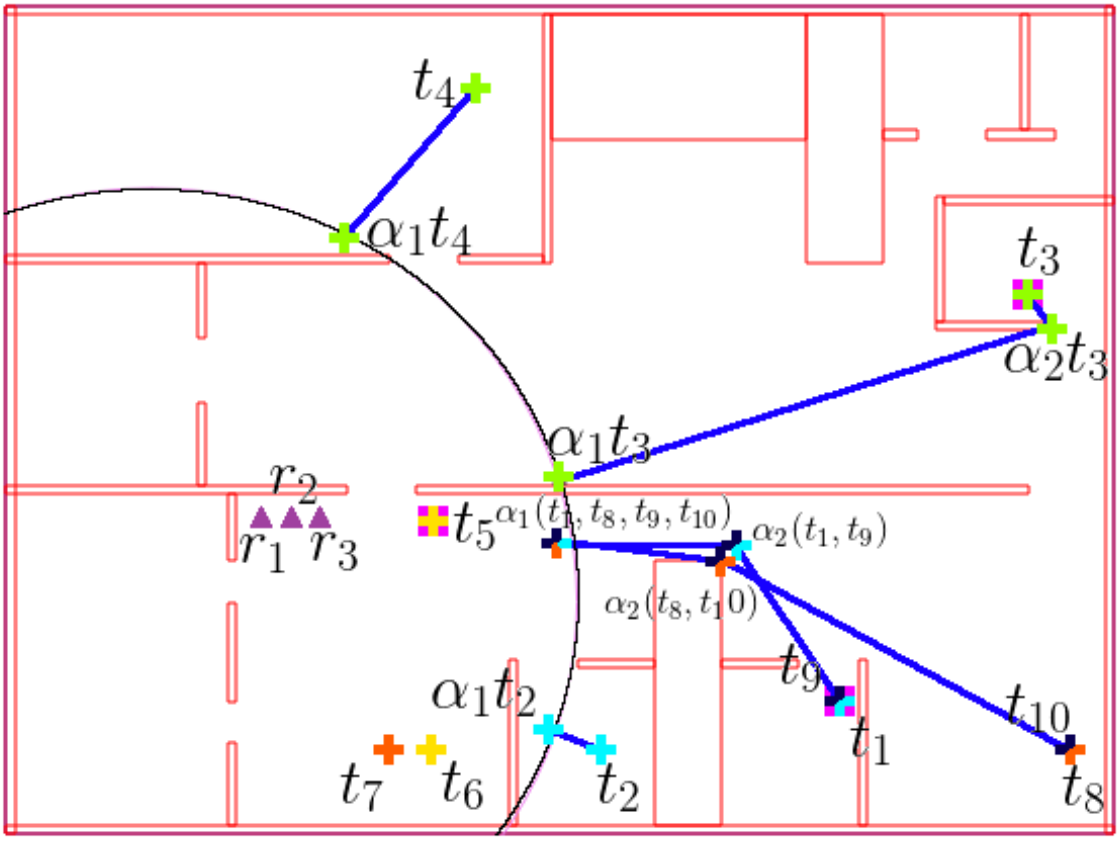
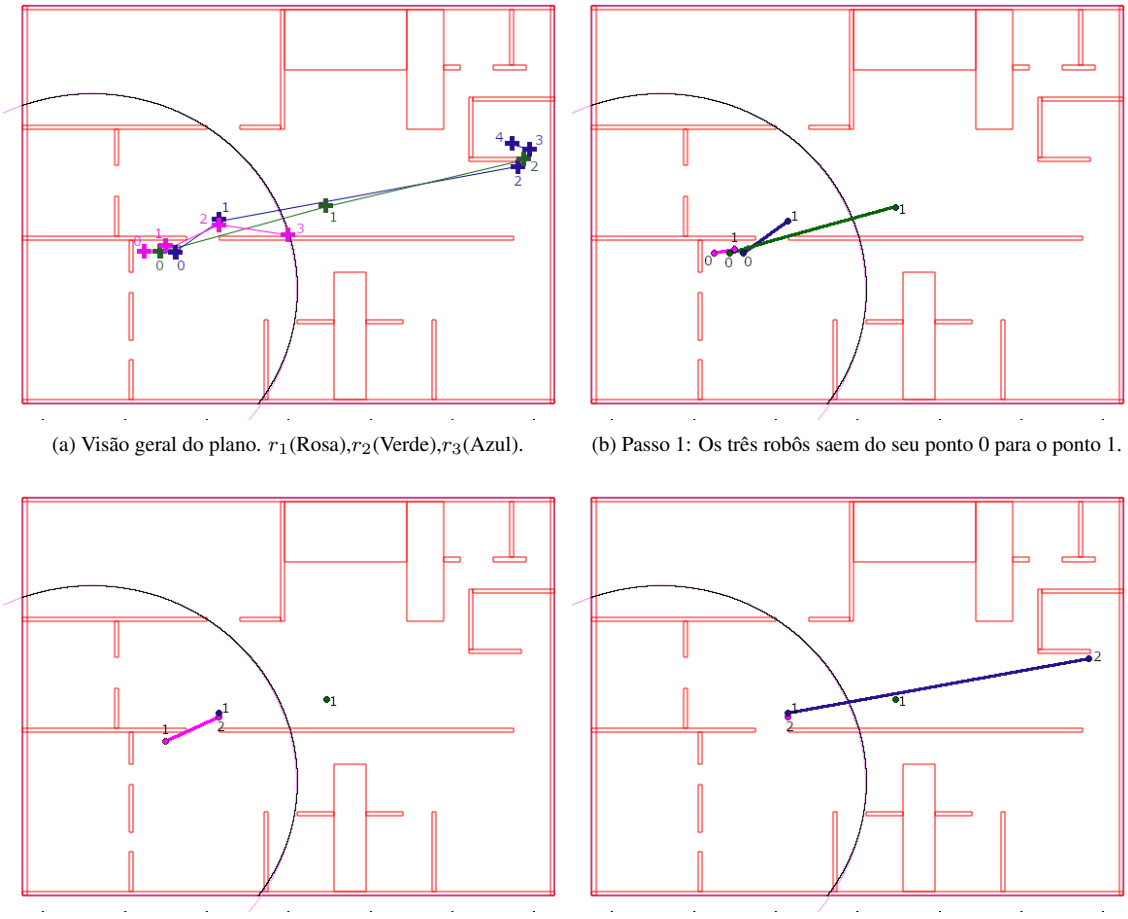


Figura 4.7: Tarefas-α para as missões atuais do SG-11

A primeira tarefa a ser executada é a tarefa t_3. O plano gerado para essa tarefa, pelo coordenador, pode ser visualizado na Figura 4.8a. Para melhor entender esse plano, recomenda-se ao leitor utilizar a Tabela 4.4. Essa tabela indica de maneira sequencial o estado em que cada um dos robôs deve estar para executar esse plano de maneira correta. Os índices dessa tabela fazem referência às posições dentro do caminho individual de cada um dos robôs, de acordo com o que está sendo mostrado na Figura 4.8a. A Figura 4.8, além do plano, também mostra os três primeiros passos de execução desse plano.

Para elucidar um pouco mais a interpretação do plano, será explicado passo a passo a sua execução. O primeiro passo, Figura 4.8b, mostra a transição do estado z_1 para o estados z_2, em que todos os robôs se deslocam de suas posições 0 para suas posições 1. O robô r_3 fornece conexão para que o robô r_2 possa se movimentar fora da zona de conexão. O segundo passo, Figura 4.8c, demonstra a transição do estado z_2 para o estado z_3, em que o robô r_1 se desloca para seu ponto 2. No terceiro passo, Figura 4.8d, ocorre a mudança de z_3 para z_4, em que o robô r_1 fornece conexão para r_2 e r_3, permitindo que r_3 saia da zona de conectividade e alcance o ponto 2 de sua trajetória. O quarto passo, Figura 4.9a, mostra a mudança do estado z_4 para z_5. Nela o robô r_2 se posiciona no seu ponto 2, que corresponde a $\alpha_2 t_3$, de forma a permitir a entrada de r_3 dentro do depósito. No quinto passo, Figura 4.9b, ocorre a transição de z_5 para z_6, o robô r_1 chega a seu destino final, a posição 3 que corresponde a $\alpha_1 t_3$. Os passos 6 e 7, Figuras $4.9c$ e 4.9d, são bem similares e mudam o estado



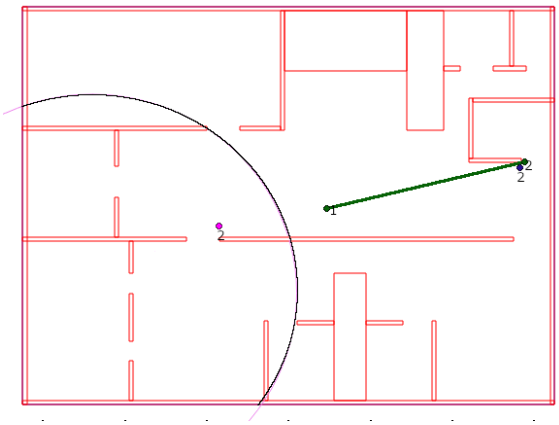
(a) Visão geral do plano. r_1(Rosa),r_2(Verde),r_3(Azul).

(b) Passo 1: Os três robôs saem do seu ponto 0 para o ponto 1.

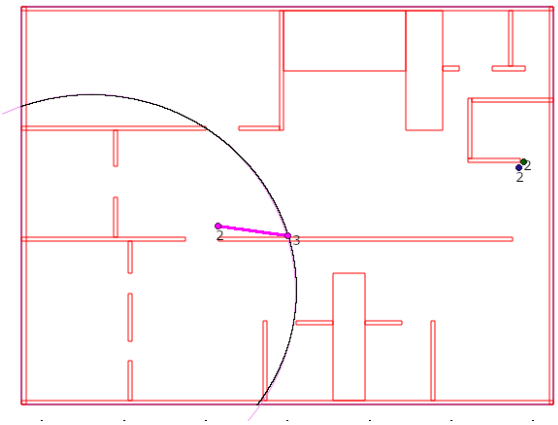
(c) Passo 2: Apenas o robô r_1 se desloca do ponto 1 ao 2.

(d) Passo 3: Apenas o robô r_3 se desloca do ponto 1 ao 2.

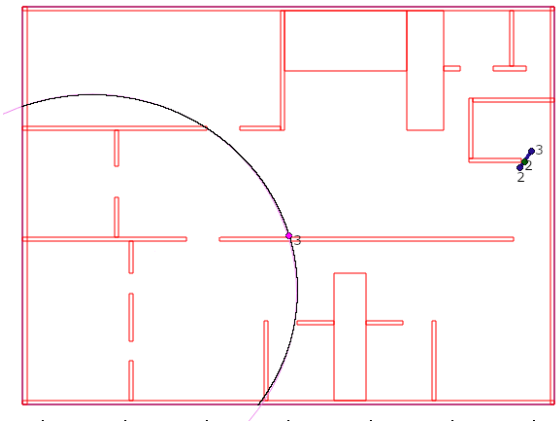
Figura 4.8: Plano gerado pelo coordenador de missão e uma visualização da execução dos três primeiros passos.



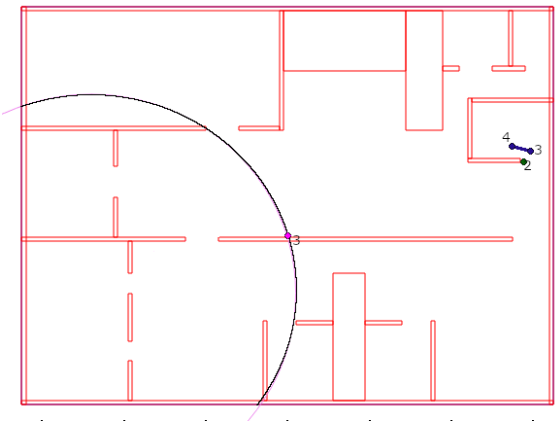
(a) Passo 4: Apenas o robô r_2 se desloca do ponto 1 ao 2.



(b) Passo 5: Robô r_1 chega ao ponto 3.



(c) Passo 6: Robô r_3 chega ao ponto 3.



(d) Passo 7: Robô r_3 chega a tarefa t_3.

Figura 4.9: Execução dos últimos quatro passos do planejamento feito pelo coordenador de missão.

Tabela 4.4: Auxílio para compreender o plano da Figura 4.8a. Cada linha representa uma mudança de estado, e o número indica em qual posição da lista de posições daquele robô ele deve estar naquele estado.

$\times$	r_1	r_2	r_3
z_1	0	0	0
z_2	1	1	1
z_3	2	1	1
z_4	2	1	2
z_5	2	2	2
z_6	3	2	2
z_7	3	2	3
z_8	3	2	4

de z_6 até z_8. Nesses passos, o robô r_3 termina seu trajeto até t_3.

Nesse passo a passo é possível visualizar que os módulos de planejamento de missão e coordenação de missão atingiram seu objetivo de fornecer um plano global de ação para cada um dos robôs de forma que eles

possam se locomover pelo ambiente sem perder a conexão com a base. Os detalhes e refinamentos dessas rotas ficam na responsabilidade do executor de movimento, que vai dispor de mais informações dinâmicas do ambiente. Essas informações vão permitir melhorar a execução desse plano e reagir a mudanças no ambiente.

O próximo passo para completar essa missão m_2 é realizar a tarefa t_4. Enquanto os robôs executam o movimento o planejador de missão já está gerando o plano do próximo passo. Só que agora os robôs devem ter como seu ponto inicial o ponto de parada da última execução. Quando o teste foi rodado pela primeira vez, ele simplesmente não terminava de calcular a rota. Ao analisar mais profundamente a causa disso, descobriu-se que existe uma falha no jeito que as tarefas-α são geradas. Como é possível observar na Figura 4.7, a tarefa-α $\alpha_1 t_4$ fica em uma região inalcançável pelos robôs se eles estiverem abaixo do cruzamento entre o obstáculo e a borda da zona de conectividade imediatamente abaixo dessa tarefa auxiliar. Para que o algoritmo conseguisse computar um trajeto, essa tarefa-α foi forçadamente trazida para baixo e então o algoritmo conseguiu rapidamente obter um resultado.

O resultado do planejamento da tarefa t_4 pode ser visualizado na Figura 4.10, com o auxílio da Tabela 4.5. Um outro ponto que indica uma falha, ou incompletude no atual modo de calcular tarefas-α, é que essa tarefa necessitou de três robôs para ser realizada, e não apenas dois como esperado. Isso ocorreu, pois o ponto de início da execução era uma configuração que necessitava de três robôs, então inevitavelmente um plano que parta desse ponto também precisará de três pontos.

Analisando o desenvolvimento desse plano é possível perceber que a única função do robô r_1 é permitir

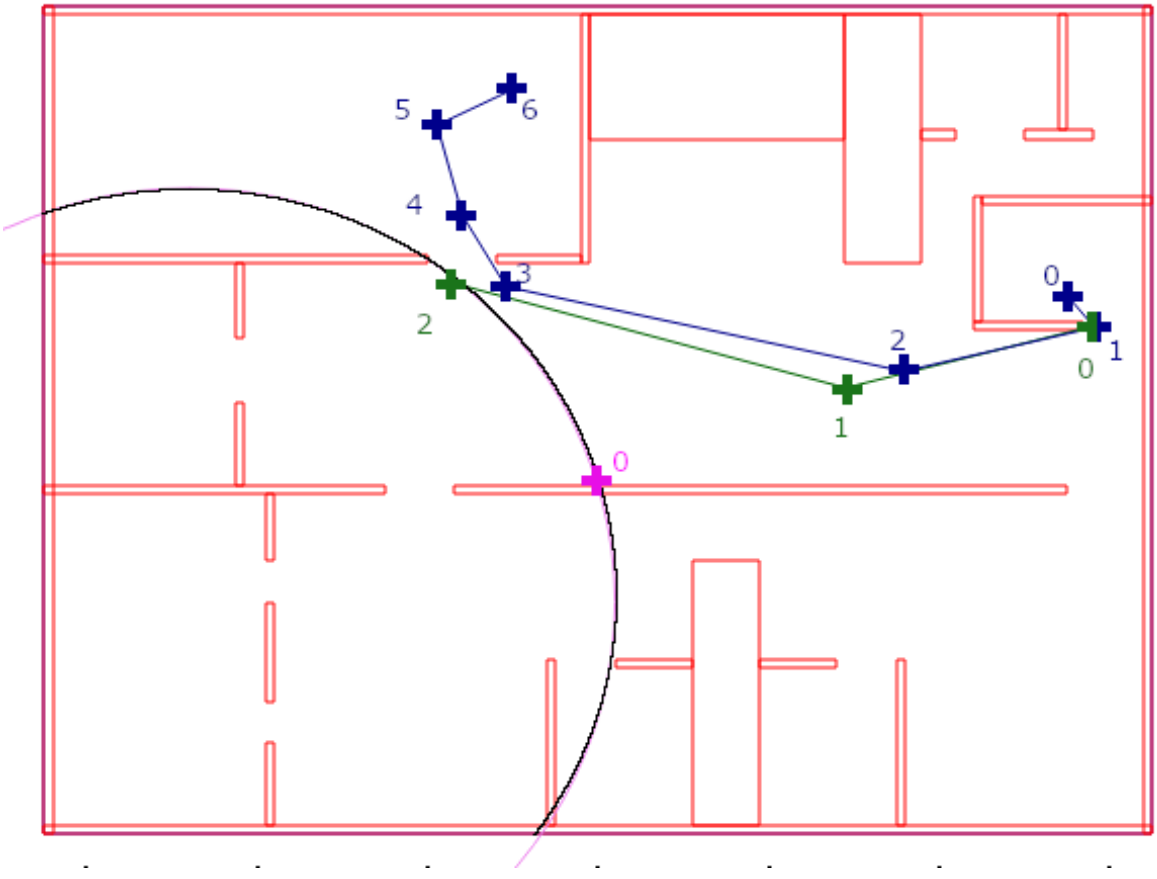


Figura 4.10: Plano de execução para a tarefa t_4.

Tabela 4.5: Auxílio para compreender o plano da Figura 4.10. Cada linha representa uma mudança de estado, e o número indica em qual posição da lista de posições daquele robô ele deve estar naquele estado.

$\times$	r_1	r_2	r_3
z_1	0	0	0
z_2	0	0	1
z_3	0	1	2
z_4	0	1	3
z_5	0	1	4
z_6	0	1	5
z_7	0	2	6

que o robô r_2 volte para a zona de conectividade, enquanto isso, o robô r_3 vai levando a carga para a sala de aula, inicialmente recebendo conexão por meio do robô r_1 e no final por meio do robô r_2.

Nesse meio tempo, o servidor já deve ter calculado qual a próxima missão a ser realizada por meio dos métodos apresentados anteriormente. O restante da simulação seria apenas repetir esses processos até que não exista mais nenhuma missão na fila de execução.

Esse teste expôs algumas falhas e incompletudes nos métodos desenvolvidos até o momento e servem como um bom ponto de partida para realizar melhorias no sistema de planejamento como um todo. Entretanto, também pode-se constatar um sucesso de conseguir gerar rotas livres de colisões físicas ou de queda de conexão que possam ser realizadas por um grupo finito de robôs. Além disso, pode-se concluir que as técnicas escolhidas servem como um bom ponto inicial para o desenvolvimento do sistema de assistente laboratorial para o LARA e para sistemas mais complexos de resgates.

5 CONCLUSÃO

5.1 CONCLUSÃO

Este trabalho apresentou uma arquitetura utilizando robôs e computação em nuvem para criar um sistema de planejamento de movimento com restrição de comunicação para a implementação de um programa de assistentes laboratoriais para o laboratório LARA. Foram desenvolvidas técnicas e ferramentas que irão servir como base para a implementação completa desse sistema. Essas ferramentas pertencem a parte não embarcada do sistema e servem como centros de planejamentos das ações do grupo de robôs, recebendo e processando as missões que são enviadas ao sistema.

Inicialmente foi realizada a apresentação de um revisão bibliográfica dos temas e trabalhos mais relacionados a este trabalho. Como por exemplo, robôs como ponto de acesso, manutenção de redes de conexão e o planejamento de movimento com restrição de comunicação. Em seguida, foi introduzida a fundamentação de conceitos básicos necessários para a compreensão do desenvolvimento das técnicas e ferramentas apresentadas. Primeiramente, se definiu e apresentou melhor os conceitos relacionados ao CAMP. Logo após, foi realizado um explicação mais detalhada de algoritmos de planejamento, começando pela apresentação dos conceitos básicos. Depois, foi definido o planejamento de movimento e o espaço de configuração. Em seguida, o funcionamento geral de planejamento foi explicado, seguido da apresentação de algoritmo de busca em árvores aleatórias como o RRT e o RRT*. Para finalizar a apresentação dos conceitos básicos, foram apresentados o algoritmo de Dijkstra, para computação de custos em grafos, e os conceitos relacionados a triangulação de Delaunay e formas-α.

Em seguida, o problema de planejamento para o sistema de assistentes laboratoriais foi detalhado e foi criada uma modelagem matemática para esse problema. Com a definição desses modelos, foi apresentada com maiores detalhes a arquitetura de planejamento de movimento com restrições de comunicação proposta. Ela é composta de três módulos principais: o planejamento de missão, a coordenação de missão e a execução do movimento. Eles dependem um do outro, mas podem funcionar de maneira paralela com um sistema de fila. Esse sistema é distribuído e os dois primeiros módulos se encontram, idealmente, em plataformas de processamento em nuvem.

O planejamento de movimento possui dois objetivos principais, gerar os pontos objetivos por onde os robôs devem passar e decidir a ordem de execução das missões e tarefas do sistema. Ele faz isso por meio das tarefas-α, que determinam quais os pontos chaves por onde os robôs devem passar e se é possível realizar uma determinada tarefa. A partir dessas tarefas auxiliares, são calculadas funções heurísticas para estimar o custo total de realização de uma tarefa e decidir qual missão deve ser executada.

A coordenação de missão é responsável por pegar os pontos objetivos gerados pelo planejador e criar um plano de execução global para cada um dos robôs. Essas trajetórias devem ser geradas de forma a sempre manter a conexão e evitar colisões com os obstáculos globalmente conhecidos. Elas são geradas utilizando algoritmos de busca em árvores aleatórias.

A execução do movimento é um módulo que está em cada um dos robôs. Ela é responsável por controlar o movimento daquele robô para que ele siga a trajetória planejada e consiga manter sua conexão e evitar obstáculos desconhecidos. Esse módulo deve ser mais reativo e utilizar os dados que ele coleta tanto para

melhorar seu planejamento local, como para enviar novas informações para que os módulos acima dele possam melhorar seus modelos e gerar planos melhores. Infelizmente, devido a restrições temporais não foi possível implementar essa parte do sistema.

Os testes realizados comprovaram que as técnicas e ferramentas desenvolvidas servem como uma boa base para o desenvolvimento do sistema de assistentes laboratoriais, e com alguns pequenos ajustes podem ser utilizados para um arcabouço mais genérico de planejamento de movimento com restrição de movimento. Os primeiros testes mostraram a capacidade que as tarefas-α possuem de serem indicadores de quantos robôs são necessários para realizar uma determinada tarefa que esteja fora de uma zona de conectividade. Os testes seguintes compararam todas as heurísticas apresentadas, indicando os benefícios e malefícios de cada uma delas, incluindo uma análise temporal de custo de execução entre essas heurísticas. Também foi analisado a escolha de qual robô deve realizar cada uma dessas tarefas para as heurísticas mais relevantes. O último teste apresentou uma simulação do ambiente do SG-11, melhor contextualizando e exemplificando o sistema desenvolvido. Apesar da identificação de alguns problemas com as ferramentas em determinadas situações. É possível constatar que o objetivo de criar um arquitetura, e desenvolver ferramentas capazes de implementar essa arquitetura, para um planejador de movimento com restrição de movimento foi alcançado com sucesso.

5.2 TRABALHO FUTUROS

Para trabalhos futuros, é proposto a criação detalhada do módulo de execução de movimento. Ele necessitaria, além das partes clássicas de detecção e evasão reativa de obstáculos, de um sistema reativo que consiga utilizar métricas de algoritmos de roteamento bem estabelecidos para manter a conexão de uma maneira mais proativa. Além disso, a implementação de um sistema de mapeamento de canais de comunicação, de forma que o planejador de missão possa gerar planos que reflitam melhor o ambiente que os robôs navegam.

Além disso, outra linha interessante de trabalho é considerar os obstáculos e o posicionamento dos robôs para o cálculo das tarefas-α, de forma a evitar a situação encontrada nos testes desenvolvidos, em que uma tarefa-α não era alcançável devido a configuração geométrica do ambiente e da zona de conectividade. Uma outra alternativa para abordar esse problema seria a criação de um quarto módulo que seria ativado nesse caso de falha, ele seria responsável por colocar os robôs em posições que eles consigam executar as tarefas e tarefas-α previamente calculados.

Por fim, um bom trabalho é a implementação desse sistema e dessas ferramentas em um conjunto de robôs reais, como os disponíveis no LARA. Isso permitirá a descoberta de novas necessidades, situações e erros que muitas vezes não aparecem em simulações, além de permitir e instigar novas pesquisas dentro do laboratório.

APÊNDICE

A EXEMPLO DE UMA MODELAGEM CAMP

Este apêndice irá mostrar um exemplo da linguagem descritiva apresentada na Seção 3.2.1. O arquivo irá descrever o primeiro andar do SG 11, em um cenário com três robôs e duas missões. A primeira missão é composta de quatro tarefas e a segunda de apenas uma missão. O resultado do processamento desse arquivo é apresentado na Figura A.1.

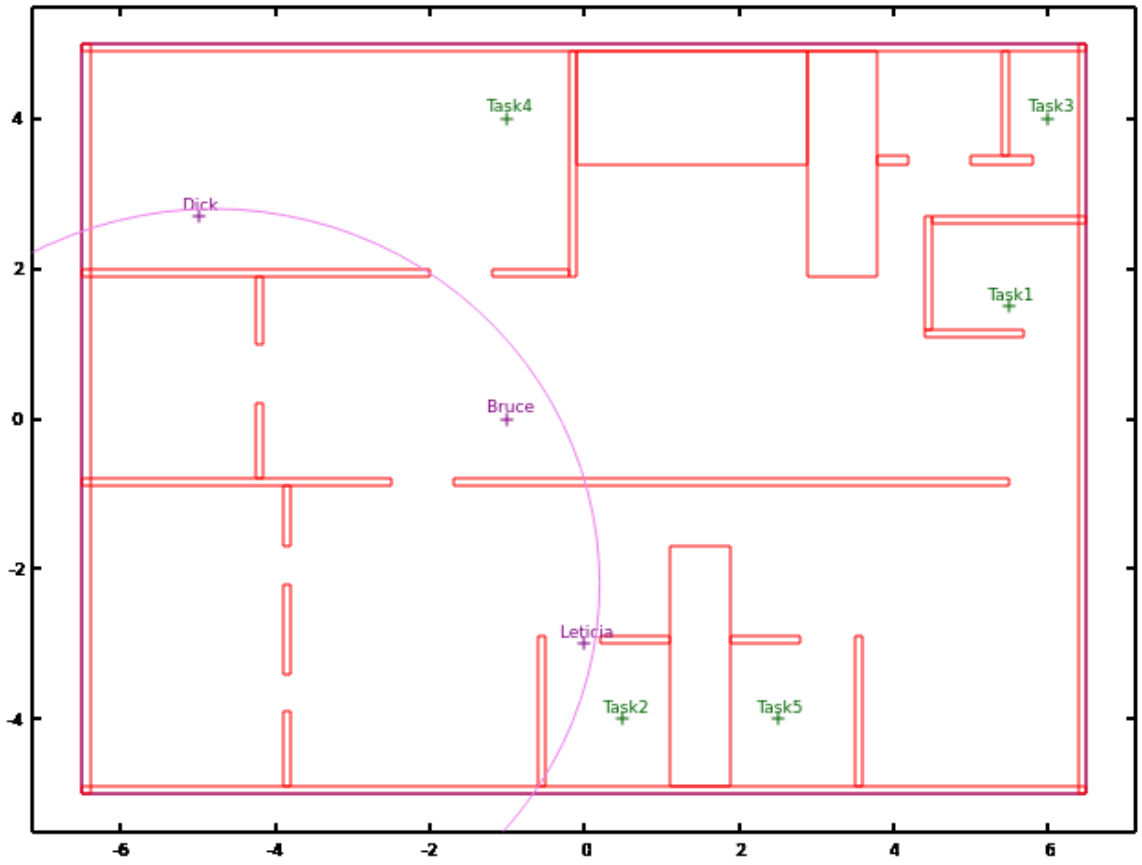


Figura A.1: Representação gráfica do arquivo descritor em JSON. A descrição retrata uma parte do primeiro andar do SG 11, em que a parte inferior serio o LARA.

ARQUIVO JSON SG11

```
{
        "robot_types":[
                "R1", "R2", "R3"
        ],
        "task_types":[
```

```
                "T1", "T2", "T3"
        ],
        "effort_function":{
                "T1":{"R1": 0.60, "R2": 0.10,            "R3": 0.34},
                "T2":{"R1": 0.43, "R2": 0.78,            "R3": 0.47},
                "T3":{"R1": 1.00, "R2":"infinity",      "R3": 0.3}
        },
        "reward_function":{
                "T1":5, "T2":2.7, "T3":0.5
        },
        "robots": [
                {
                        "name": "Bruce",
                        "type": "R2",
                        "position": {
                                "type": "2d",
                                "x": -1.0,
                                "y": 0.0
                        }
                },
                {
                        "name": "Dick",
                        "type": "R3",
                        "position": {
                                "type": "2d",
                                "x": -5.0,
                                "y": 2.7
                        }
                },
                {
                        "name": "Leticia",
                        "type": "R1",
                        "position": {
                                "type": "2d",
                                "x": 0,
                                "y": -3
                        }
                }
        ],
        "tasks":[
                {
                        "id":"Task1",
                        "type":"T1",
                        "position":{
                                "type":"2d",
```

```
                                "x":5.5,
                                "y":1.5
                        },
                        "prerequisites":["Task2"]
                },
                {
                        "id":"Task2",
                        "type":"T2",
                        "position":{
                                "type":"2d",
                                "x":0.5,
                                "y":-4.0
                        },
                        "prerequisites":["Task3", "Task4"]
                },
                {
                        "id":"Task3",
                        "type":"T1",
                        "position":{
                                "type":"2d",
                                "x":6.0,
                                "y":4.0
                        },
                        "prerequisites":[]
                },
                {
                        "id":"Task4",
                        "type":"T3",
                        "position":{
                                "type":"2d",
                                "x":-1.0,
                                "y":4.0
                        },
                        "prerequisites":[]
                },
                {
                        "id":"Task5",
                        "type":"T2",
                        "position":{
                                "type":"2d",
                                "x":2.5,
                                "y":-4.0
                        },
            "prerequisites":[]
                }
```

```
        ],
        "missions": [
                {
                        "id":"Mission1",
                        "priority":5,
                        "deadline":120,
                        "tasks": [
                                "Task1", "Task2", "Task3", "Task4"
                        ]
                },
                {
                        "id":"Mission2",
                        "priority":4,
                        "deadline":80,
                        "tasks":[
                                "Task5"
                        ]
                }
        ],
        "search_area":{
                "type":"rectangle",
                "bottom_left": {
                        "type":"2d",
                        "x":-6.5,
                        "y":-5
                },
                "width":13,
                "height":10
        },
        "obstructed_area":{
                "type":"geometric",
                "areas":[
                        {
                                "type":"rectangle",
                                "bottom_left": {
                                        "type":"2d",
                                        "x":-6.5,
                                        "y":-5
                                },
                                "width":13,
                                "height":0.1
                        },
                        {
                                "type":"rectangle",
```

```
                "top_left": {
                        "type":"2d",
                                "x":-6.5,
                                "y":5
                },
                "width":13,
                "height":0.1
        },
        {
                "type":"rectangle",
                "top_left": {
                        "type":"2d",
                        "x":-6.5,
                        "y":5
                },
                "width":0.1,
                "height":10
        },
        {
                "type":"rectangle",
                "top_right": {
                        "type":"2d",
                        "x":6.5,
                        "y":5
                },
                "width":0.1,
                "height":10
        },
        {
                "type":"rectangle",
                "top_left": {
                        "type":"2d",
                        "x":-6.5,
                        "y":2
                },
                "width":4.5,
                "height":0.1
        },
        {
                "type":"rectangle",
                "top_left": {
                        "type":"2d",
                        "x":-1.2,
                        "y":2
                },
```

```
                        "width":1.0,
                        "height":0.1
                },
                {
                        "type":"rectangle",
                        "bottom_left": {
                                "type":"2d",
                                "x":-0.2,
                                "y":1.9
                        },
                        "width":0.1,
                        "height":3.0
                },
                {
                        "type":"rectangle",
                        "bottom_left": {
                                "type":"2d",
                                "x":-0.1,
                                "y":3.4
                        },
                        "width":3,
                        "height":1.5
                },
                {
                        "type":"rectangle",
                        "bottom_left": {
                                "type":"2d",
                                "x":2.9,
                                "y":1.9
                        },
                        "width":0.9,
                        "height":3
                },
                {
                        "type":"rectangle",
                        "bottom_left": {
                                "type":"2d",
                                "x":3.8,
                                "y":3.4
                        },
                        "width":0.4,
                        "height":0.1
                },
                {
                        "type":"rectangle",
```

```
                "bottom_left": {
                        "type":"2d",
                        "x":5,
                        "y":3.4
                },
                "width":0.8,
                "height":0.1
        },
        {
                "type":"rectangle",
                "bottom_left": {
                        "type":"2d",
                        "x":5.4,
                        "y":3.5
                },
                "width":0.1,
                "height":1.4
        },
        {
                "type":"rectangle",
                "top_right": {
                        "type":"2d",
                        "x":6.5,
                        "y":2.7
                },
                "width":2.0,
                "height":0.1
        },
        {
                "type":"rectangle",
                "top_right": {
                        "type":"2d",
                        "x":4.5,
                        "y":2.7
                },
                "width":0.1,
                "height":1.5
        },
        {
                "type":"rectangle",
                "top_left": {
                        "type":"2d",
                        "x":4.4,
                        "y":1.2
                },
```

```
                "width":1.3,
                "height":0.1
            },
            {
                "type":"rectangle",
                "top_left": {
                    "type":"2d",
                    "x":-4.25,
                    "y":1.9
                },
                "width":0.1,
                "height":0.9
            },
            {
                "type":"rectangle",
                "top_left": {
                    "type":"2d",
                    "x":-4.25,
                    "y":0.2
                },
                "width":0.1,
                "height":1.0
            },
            {
                "type":"rectangle",
                "top_left": {
                    "type":"2d",
                    "x":-6.5,
                    "y":-0.8
                },
                "width":4.0,
                "height":0.1
            },
            {
                "type":"rectangle",
                "top_left": {
                    "type":"2d",
                    "x":-1.7,
                    "y":-0.8
                },
                "width":7.2,
                "height":0.1
            },
            {
                "type":"rectangle",
```

```
                        "bottom_left": {
                                "type":"2d",
                                "x":-3.9,
                                "y":-4.9
                        },
                        "width":0.1,
                        "height":1.0
                },
                {
                        "type":"rectangle",
                        "bottom_left": {
                                "type":"2d",
                                "x":-3.9,
                                "y":-3.4
                        },
                        "width":0.1,
                        "height":1.2
                },
                {
                        "type":"rectangle",
                        "bottom_left": {
                                "type":"2d",
                                "x":-3.9,
                                "y":-1.7
                        },
                        "width":0.1,
                        "height":0.8
                },
                {
                        "type":"rectangle",
                        "bottom_left": {
                                "type":"2d",
                                "x":-0.6,
                                "y":-4.9
                        },
                        "width":0.1,
                        "height":2.0
                },
                {
                        "type":"rectangle",
                        "bottom_left": {
                                "type":"2d",
                                "x":0.2,
                                "y":-3.0
                        },
```

```
                                "width":0.9,
                                "height":0.1
                        },
                        {
                                "type":"rectangle",
                                "bottom_left": {
                                        "type":"2d",
                                        "x":3.5,
                                        "y":-4.9
                                },
                                "width":0.1,
                                "height":2.0
                        },
                        {
                                "type":"rectangle",
                                "bottom_left": {
                                        "type":"2d",
                                        "x":1.9,
                                        "y":-3.0
                                },
                                "width":0.9,
                                "height":0.1
                        },
                        {
                                "type":"rectangle",
                                "center": {
                                        "type":"2d",
                                        "x":1.5,
                                        "y":-3.3
                                },
                                "width":0.8,
                                "height":3.2
                        }
                ]
        },
        "connectivity_function":{
                "type":"geometric",
                "areas":[
                        {
                                "type":"circle",
                                "center": {
                                        "type":"2d",
                                        "x":-4.8,
                                        "y":-2.2
                                },
```

```
                                    "radius":5
                                }
                    ]
            }
}
```

[1] J. Baruch and M. J. Cox, "Remote control and robots: an internet solution," in *Fuel and Energy Abstracts*, vol. 6, p. 434, 1996.

[2] K. Goldberg, S. Gentner, C. Sutter, and J. Wiegley, "The mercury project: A feasibility study for internet robots," *IEEE Robotics & Automation Magazine*, vol. 7, no. 1, pp. 35–40, 2000.

[3] K. Taylor and B. Dalton, "Internet robots: A new robotics niche," *IEEE Robotics & Automation Magazine*, vol. 7, no. 1, pp. 27–34, 2000.

[4] K. Katsaros and M. Dianati, "A conceptual 5g vehicular networking architecture," in *5G Mobile Communications*, pp. 595–623, Springer, 2017.

[5] S. Pandit, F. H. Fitzek, C. Lehmann, D. Nophut, D. Kiss, V. Kovacs, A. Nagy, G. Csorvasi, M. Tóth, T. Rajacsis, *et al.*, "Joint design of communication and control for connected cars in 5g communication systems," in *2016 IEEE Globecom Workshops (GC Wkshps)*, pp. 1–7, IEEE, 2016.

[6] S. Pandit, F. H. Fitzek, and S. Redana, "Demonstration of 5g connected cars," in *2017 14th IEEE Annual Consumer Communications & Networking Conference (CCNC)*, pp. 605–606, IEEE, 2017.

[7] P. P. Ray, "Internet of robotic things: Concept, technologies, and challenges," *IEEE Access*, vol. 4, pp. 9489–9500, 2016.

[8] A. Ghaffarkhah and Y. Mostofi, "Communication-aware motion planning in mobile networks," *IEEE Transactions on Automatic Control*, vol. 56, no. 10, pp. 2478–2485, 2011.

[9] A. Khasawneh, H. Rogers, J. Bertrand, K. C. Madathil, and A. Gramopadhye, "Human adaptation to latency in teleoperated multi-robot human-agent search and rescue teams," *Automation in Construction*, vol. 99, pp. 265–277, 2019.

[10] M. Wzorek, C. Berger, P. Rudol, and P. Doherty, "Deployment of ad hoc network nodes using uavs for search and rescue missions," in *2018 International Electrical Engineering Congress (iEECON)*, pp. 1–4, IEEE, 2018.

[11] J. Delmerico, S. Mintchev, A. Giusti, B. Gromov, K. Melo, T. Horvat, C. Cadena, M. Hutter, A. Ijspeert, D. Floreano, *et al.*, "The current state and future outlook of rescue robotics," *Journal of Field Robotics*, vol. 36, no. 7, pp. 1171–1191, 2019.

[12] R. W. d. S. M. d. Oliveira, "Uma arquitetura de navegação para robôs móveis," 2017.

[13] L. H. Silva Porto, R. Werberich da Silva Moreira de Oliveira, R. Bauchspiess, C. Brito, L. F. da Cruz Figueredo, G. Araujo Borges, and G. Novaes Ramos, "An autonomous mobile robot architecture for outdoor competitions," in *2018 Latin American Robotic Symposium, 2018 Brazilian Symposium on Robotics (SBR) and 2018 Workshop on Robotics in Education (WRE)*, pp. 141–146, Nov 2018.

[14] M. A. Batalin and G. S. Sukhatme, "Coverage, exploration and deployment by a mobile robot and communication network," *Telecommunication Systems*, vol. 26, no. 2-4, pp. 181–196, 2004.

[15] M. A. Vieira, R. Govindan, and G. S. Sukhatme, "Towards autonomous wireless backbone deployment in highly-obstructed environments," in *2011 IEEE International Conference on Robotics and Automation*, pp. 5369–5374, IEEE, 2011.

[16] M. A. Vieira, R. Govindan, and G. S. Sukhatme, "An autonomous wireless networked robotics system for backbone deployment in highly-obstructed environments," *Ad Hoc Networks*, vol. 11, no. 7, pp. 1963–1974, 2013.

[17] E. R. Santos and M. A. Vieira, "Autonomous wireless backbone deployment with bounded number of networked robots," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3740–3746, IEEE, 2014.

[18] S. K. Pandey and M. A. Zaveri, "Quasi random deployment and localization in layered framework for the internet of things," *The Computer Journal*, vol. 61, no. 2, pp. 159–179, 2017.

[19] A. H. Mong-ying, A. Cowley, V. Kumar, and C. J. Taylor, "Towards the deployment of a mobile robot network with end-to-end performance guarantees," in *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pp. 2085–2090, IEEE, 2006.

[20] F. Zeiger, N. Kraemer, M. Sauer, and K. Schilling, "Challenges in realizing ad-hoc networks based on wireless lan with mobile robots," in *2008 6th International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks and Workshops*, pp. 632–639, IEEE, 2008.

[21] F. Zeiger, N. Kraemer, and K. Schilling, "Commanding mobile robots via wireless ad-hoc networks - a comparison of four ad-hoc routing protocol implementations," in *2008 IEEE International Conference on Robotics and Automation*, pp. 590–595, IEEE, 2008.

[22] H. Jiang, Y. Q. Huang, and G. Y. Yang, "Formation and network chain control in sparse robot ad hoc networks based on potential field," in *2008 International Conference on Networking, Architecture, and Storage*, pp. 147–148, IEEE, 2008.

[23] E. Budianto, A. Hafidh, F. Al Afif, A. Wibowo, W. Jatmiko, B. Hardian, P. Mursanto, and A. Muis, "Autonomous telecommunication networks coverage area expansion in disaster area using mobile robots," in *2011 International Symposium on Micro-NanoMechatronics and Human Science*, pp. 361–366, IEEE, 2011.

[24] X. Zhong and Y. Zhou, "Establishing and maintaining wireless communication coverage among multiple mobile robots using artificial neural network," in *2011 IEEE International Conference on Robotics and Biomimetics*, pp. 2083–2089, IEEE, 2011.

[25] X. Zhong and Y. Zhou, "Maintaining wireless communication coverage among multiple mobile robots using fuzzy neural network," in *Proceedings of 2012 IEEE/ASME 8th IEEE/ASME International Conference on Mechatronic and Embedded Systems and Applications*, pp. 35–41, IEEE, 2012.

[26] N. Chatzipanagiotis and M. M. Zavlanos, "Distributed scheduling of network connectivity using mobile access point robots," *IEEE Transactions on Robotics*, vol. 32, no. 6, pp. 1333–1346, 2016.

[27] A. Bader and M.-S. Alouini, "Mobile ad hoc networks in bandwidth-demanding mission-critical applications: Practical implementation insights," *IEEE Access*, vol. 5, pp. 891–910, 2016.

[28] A. Ghaffarkhah and Y. Mostofi, "Path planning for networked robotic surveillance," *IEEE Transactions on Signal Processing*, vol. 60, no. 7, pp. 3560–3575, 2012.

[29] Y. Yan and Y. Mostofi, "Co-optimization of communication and motion planning of a robotic operation under resource constraints and in fading environments," *IEEE Transactions on Wireless Communications*, vol. 12, no. 4, pp. 1562–1572, 2013.

[30] J. Fink, A. Ribeiro, and V. Kumar, "Robust control for mobility and wireless communication in cyber–physical systems with application to robot teams," *Proceedings of the IEEE*, vol. 100, no. 1, pp. 164–178, 2011.

[31] J. Fink, A. Ribeiro, and V. Kumar, "Motion planning for robust wireless networking," in *2012 IEEE International Conference on Robotics and Automation*, pp. 2419–2426, IEEE, 2012.

[32] J. Fink, A. Ribeiro, and V. Kumar, "Robust control of mobility and communications in autonomous robot teams," *IEEE Access*, vol. 1, pp. 290–309, 2013.

[33] Y. Yan and Y. Mostofi, "To go or not to go: On energy-aware and communication-aware robotic operation," *IEEE Transactions on Control of Network Systems*, vol. 1, no. 3, pp. 218–231, 2014.

[34] U. Ali, Y. Yan, Y. Mostofi, and Y. Wardi, "An optimal control approach for communication and motion co-optimization in realistic fading environments," in *2015 American Control Conference (ACC)*, pp. 2930–2935, IEEE, 2015.

[35] Y. Kantaros and M. M. Zavlanos, "Distributed communication-aware coverage control by mobile sensor networks," *Automatica*, vol. 63, pp. 209–220, 2016.

[36] U. Ali, H. Cai, Y. Mostofi, and Y. Wardi, "Motion and communication co-optimization with path planning and online channel prediction," in *2016 American Control Conference (ACC)*, pp. 7079–7084, IEEE, 2016.

[37] V. Ramaswamy, S. Moon, E. W. Frew, and N. Ahmed, "Mutual information based communication aware path planning: A game theoretic perspective," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1823–1828, IEEE, 2016.

[38] Y. Marchukov and L. Montano, "Communication-aware planning for robot teams deployment," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 6875–6881, 2017.

[39] Y. Kantaros and M. M. Zavlanos, "Global planning for multi-robot communication networks in complex environments," *IEEE Transactions on Robotics*, vol. 32, no. 5, pp. 1045–1061, 2016.

[40] Y. Kantaros and M. M. Zavlanos, "Distributed intermittent connectivity control of mobile robot networks," *IEEE Transactions on Automatic Control*, vol. 62, no. 7, pp. 3109–3121, 2016.

[41] S. Caccamo, R. Parasuraman, L. Freda, M. Gianni, and P. Ögren, "Rcamp: A resilient communication-aware motion planner for mobile robots with autonomous repair of wireless connectivity," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2010–2017, IEEE, 2017.

[42] M. M. LINDHÉ, "*Communication Aware motion planning for mobile robots*," 2012.

[43] S. M. LaValle, *Planning algorithms*. Cambridge university press, 2006.

[44] M. Mohanan and A. Salgoankar, "A survey of robotic motion planning in dynamic environments," *Robotics and Autonomous Systems*, vol. 100, pp. 171–185, 2018.

[45] R. A. Conn and M. Kam, "Robot motion planning on n-dimensional star worlds among moving obstacles," *IEEE Transactions on Robotics and Automation*, vol. 14, no. 2, pp. 320–325, 1998.

[46] T. Lozano-Perez, "Spatial planning: A configuration space approach," in *Autonomous robot vehicles*, pp. 259–271, Springer, 1990.

[47] C. Fulgenzi, A. Spalanzani, and C. Laugier, "Probabilistic motion planning among moving obstacles following typical motion patterns," in *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 4027–4033, IEEE, 2009.

[48] S. R. Lindemann and S. M. LaValle, "Current issues in sampling-based motion planning," in *Robotics Research. The Eleventh International Symposium*, pp. 36–54, Springer, 2005.

[49] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[50] I. Noreen, A. Khan, and Z. Habib, "A comparison of rrt, rrt* and rrt*-smart path planning algorithms," *International Journal of Computer Science and Network Security (IJCSNS)*, vol. 16, no. 10, p. 20, 2016.

[51] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," 1998.

[52] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The international journal of robotics research*, vol. 30, no. 7, pp. 846–894, 2011.

[53] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[54] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[55] D.-T. Lee and B. J. Schachter, "Two algorithms for constructing a delaunay triangulation," *International Journal of Computer & Information Sciences*, vol. 9, no. 3, pp. 219–242, 1980.

[56] B. Delaunay *et al.*, "Sur la sphere vide," *Izv. Akad. Nauk SSSR, Otdelenie Matematicheskii i Estestvennyka Nauk*, vol. 7, no. 793-800, pp. 1–2, 1934.

[57] G. Voronoi, "Nouvelles applications des paramètres continus à théorie des formes quadratiques. deuxième mémoire. recherches sur les parallélоèdres primitifs.," *Journal für die reine und angewandte Mathematik (Crelles Journal)*, vol. 1909, no. 136, pp. 67–182, 1909.

[58] H. Edelsbrunner, D. Kirkpatrick, and R. Seidel, "On the shape of a set of points in the plane," *IEEE Transactions on information theory*, vol. 29, no. 4, pp. 551–559, 1983.

[59] F. Bernardini and C. L. Bajaj, "Sampling and reconstructing manifolds using alpha-shapes," 1997.

[60] H. Edelsbrunner, *Weighted alpha shapes*, vol. 92. University of Illinois at Urbana-Champaign, Department of Computer Science, 1992.

[61] H. Edelsbrunner and E. P. Mücke, "Three-dimensional alpha shapes," *ACM Transactions on Graphics (TOG)*, vol. 13, no. 1, pp. 43–72, 1994.

[62] M. A. Hsieh, A. Cowley, V. Kumar, and C. J. Taylor, "Maintaining network connectivity and performance in robot teams," *Journal of field robotics*, vol. 25, no. 1-2, pp. 111–131, 2008.

[63] J. H. Conway and N. J. A. Sloane, *Sphere packings, lattices and groups*, vol. 290. Springer Science & Business Media, 2013.