Universidade Federal
do Rio de Janeiro
Escola Politécnica

PLANEJAMENTO DE TRAJETÓRIA DE ROBÔS MÓVEIS EM AMBIENTES ESTRUTURADOS UTILIZANDO AUTÔMATOS E ÁLGEBRA MIN-PLUS

Antonio Galiza Cerdeira Gonzalez

Projeto de Graduação apresentado ao Corpo Docente do Curso de Engenharia de Controle e Automação da Escola Politécnica da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Engenheiro de Controle e Automação.

Orientadores: Marcos Vinícius Silva Alves
Lilian Kawakami Carvalho

Rio de Janeiro
Novembro de 2016

PLANEJAMENTO DE TRAJETÓRIA DE ROBÔS MÓVEIS EM AMBIENTES ESTRUTURADOS UTILIZANDO AUTÔMATOS E ÁLGEBRA MIN-PLUS

Antonio Galiza Cerdeira Gonzalez

PROJETO DE GRADUAÇÃO SUBMETIDO AO CORPO DOCENTE DO CURSO DE ENGENHARIA DE CONTROLE E AUTOMAÇÃO DA ESCOLA POLITÉCNICA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE ENGENHEIRO DE CONTROLE E AUTOMAÇÃO.

Examinado por:

Prof. Marcos Vinícius Silva Alves, M.Sc.

Prof. Lilian Kawakami Carvalho, D.Sc.

Prof. Gustavo da Silva Viana , M.Sc

Prof. Marcos Vicente de Brito Moreira, D.Sc.

RIO DE JANEIRO, RJ – BRASIL
NOVEMBRO DE 2016

Galiza Cerdeira Gonzalez, Antonio

PLANEJAMENTO DE TRAJETÓRIA DE ROBÔS MÓVEIS EM AMBIENTES ESTRUTURADOS UTILIZANDO AUTÔMATOS E ÁLGEBRA MIN-PLUS / Antonio Galiza Cerdeira Gonzalez. – Rio de Janeiro: UFRJ/Escola Politécnica, 2016.

VI, 51 p.: il.; 29, 7cm.

Orientadores: Marcos Vinícius Silva Alves
Lilian Kawakami Carvalho

Projeto de Graduação – UFRJ/Escola Politécnica/ Curso de Engenharia de Controle e Automação, 2016.

Referências Bibliográficas: p. 51 – 51.

1. Sistemas a eventos discretos. 2. Autõmatos. 3. Robótica Móvel. 4. Planejamento de Trajetória. 5. Álgebra *min-plus*. I. Silva Alves, Marcos Vinícius *et al.* II. Universidade Federal do Rio de Janeiro, Escola Politécnica, Curso de Engenharia de Controle e Automação. III. PLANEJAMENTO DE TRAJETÓRIA DE ROBÔS MÓVEIS EM AMBIENTES ESTRUTURADOS UTILIZANDO AUTÔMATOS E ÁLGEBRA MIN-PLUS.

A todos aqueles a quem sempre
hei de amar, tanto na Terra,
quanto no Céu.

Agradecimentos

Gostaria de agradecer primeiro a Deus Pai todo poderoso, sem o qual nada seria possível. *Non nobis, Domine, non nobis, sed nomini tuo da gloriam.*

Aos meus pais, José e Elisiane, por todo amor, carinho, dedicação e lições ao longo da vida; sem os quais nada seria possível. Amo-os de todo o coração; e espero sempre fazê-los felizes.

Ao meu irmão Lucas, por todo companheirismo e amizade e por todo apoio e compreensão durante minha a graduação.

Aos meus avós, Maria (*in memoriam*), José, Elizabeth e Antônio; por todo carinho, lições de vida e tantos bons momentos que jamais hei de esquecer.

A José Gomes Pinto, o tão querido Dedé, a quem considero em grande estima por todo o carinho e dedicação dedicados à nossa família; que fazem os dias que passamos em Fortaleza mais felizes.

Aos meus tios e tias, Maricarmem, Gentil, Christine, Antônio e Vera, por todo carinho e apoio ao longo da vida; bem como aos meus primos e primas Felipe, Daniela, André, Lilian, Tiago e Bárbara; por toda amizade.

Aos meus professores, que tanto me ensinaram e que tornaram possível realizar este sonho que é a graduação.

Aos meus orientadores, Lilian Kawakami e Marcos Vinícius, que por tanto tempo se dedicaram a me apoiar e a motivar com sua atenção e amizade; além de contribuírem enormemente para a realização deste trabalho.

Ao meu orientador da iniciação científica, João Carlos do Santos Basilio, por tanta dedicação e amizade demonstrada ao longo dos anos que passei sob sua orientação.

Aos colegas e amigos do Laboratório de Controle e Automação, Eduardo Nunes, Gustavo Viana, Felipe Cabral, Ingrid Antunes; por todo o apoio e amizade

Aos meus amigos do curso de Engenharia de Controle e Automação, que sempre me apoiaram, animaram e ajudaram a concluir a faculdade; José Guilherme, André Hwang, Pedro Gomes, Raphael Parreira, Gabriel Lira, Vinícius Plácido, Tiago Azevedo, Ricardo Oliveira, Gabriel Ribeiro, Lívia Paravidino, Isabelle Contreras, Derek Chan, Gabriel Evangelista, Larissa de Oliveira, Daniel Ramos, Diego Gonzalez, Rob Kler Júnior, Bruno Valério, Eduardo Brandão, Thiago Gláuber, João Vitor Pércia,

Jonas Karst, Thiago Piñeiro, Dan Rodrigues, Sidney Kolás, Felipe Matheus, Rodrigo Macedo, Guilherme Avelino, Renan Calmon, Gabriel Eiras, Pedro Henrique, Gabriel Loureiro, Isabela Quintanilha, Luís Gustavo, Alexandre Alves, Fernanda Folly, Cayo Valsamis e João Gonçalves.

Ao professor Daniel Villela, que muito ensinou não só em termos de autodefesa, mas em diversas questões da vida e da faculdade; por todo seu carinho, apoio e amizade.

Aos grandes amigos que fiz na faculdade, por toda a companhia e apoio durante a graduação; Henrique Maia, Rafael Prallon, Gabriel De Jong, Juliana Christina, Erick Gama, Yangye He, Matheus Soares, Aloizio Macedo, Daphne Poll e André Segadas.

Aos bons amigos Gustavo Amaral, Marcos Paulo, João Pedro e João Vitor, companheiros de tantas aventuras e conversas animadas.

Ao CNPq, pela bolsa de Iniciação Científica que muito me ajudou ao longo da graduação e que apoiou a criação da base deste projeto de graduação.

Por fim, gostaria de agradecer a todos os brasileiros, cujos esforços coletivos custeiam a instituição de excelência que é a UFRJ.

Resumo do Projeto de Graduação apresentado à Escola Politécnica/UFRJ como parte dos requisitos necessários para a obtenção do grau de Engenheiro de Controle e Automação

PLANEJAMENTO DE TRAJETÓRIA DE ROBÔS MÓVEIS EM AMBIENTES ESTRUTURADOS UTILIZANDO AUTÔMATOS E ÁLGEBRA MIN-PLUS

Antonio Galiza Cerdeira Gonzalez

Novembro/2016

Orientadores: Marcos Vinícius Silva Alves
Lilian Kawakami Carvalho

Curso: Engenharia de Controle e Automação

O emprego de robôs em ambientes industriais significou uma verdadeira revolução na capacidade produtiva humana, tanto em termos de qualidade quanto de quantidade; tornado-se um dos elementos mais impactantes da 3ª revolução industrial. Neste trabalho, consideramos o problema em que um robô trafega em um armazém industrial com a tarefa de transportar objetos de um ponto de carga e descarga até uma das prateleiras do armazém e vice e versa. Para tanto, desenvolvemos um sistema de planejamento de trajetória e navegação em um ambiente industrial estruturado com possíveis obstáculos desconhecidos *a priori*. Na abordagem proposta, modelamos o ambiente estruturado por meio de um autômato e utilizamos a álgebra *min-plus* matricial para implementar o algoritmo de Floyd-Warshall com objetivo de calcular o menor caminho entre o ponto de carga e descarga e o ponto de destino, desviando de possíveis obstáculos desconhecidos *a priori*. O sistema de planejamento e navegação proposto foi implementado em uma plataforma real *Pioneer P3DX*.

Abstract of Graduation Project presented to POLI/UFRJ as a partial fulfillment of the requirements for the degree of Control and Automation Engineer

TRAJECTORY PLANNING FOR MOBILE ROBOTS IN STRUCTURED AMBIENTS USING AUTOMATA

Antonio Galiza Cerdeira Gonzalez

November/2016

Advisors: Marcos Vinícius Silva Alves
Lilian Kawakami Carvalho

Course: Control and Automation Engineering

The use of robots in industrial environments meant a revolution in human productive capacity, both in terms of quality and quantity; making it one of the most striking elements of the 3^{rd} industrial revolution. In this work, we consider the problem in which a robot travels in an industrial warehouse with the task of transporting objects from a loading and unloading point to one of the warehouse shelves and vice versa. Therefore, we developed a path planning and navigation system for a structured industrial environment with possible *a priori* unknown obstacles. In the proposed approach, we model the structured environment by an automaton and use *min-plus* matrix algebra to implement Floyd-Warshall algorithm in order to calculate the shortest path between the point of loading and unloading and the destination point, detouring the *a priori* unknown obstacles. The proposed planning and navigation systems were implemented in a real platform *Pioneer P3DX*.

Sumário

Lista de Figuras **iv**

Lista de Tabelas **vi**

1 Introdução **1**

2 Fundamentos Teóricos **3**

2.1 Sistemas a Eventos Discretos 3

2.1.1 Linguagem 4

2.1.2 Autômatos 5

2.2 Álgebra min-plus 6

2.3 Algoritmos de busca do Menor Caminho 8

2.3.1 Algoritmo de Dijkstra 8

2.3.2 Algoritmo de Floyd-Warshall 12

3 Controle de Navegação **16**

3.1 Introdução 16

3.2 Modelagem 16

3.3 Projeto do Controlador 20

4 Planejamento de Trajetória **25**

4.1 Introdução 25

4.2 Modelagem do ambiente utilizando autômato 25

4.3 Algoritmos de planejamento de trajetória 32

4.4 Implementação 36

4.4.1 Ambiente sem obstáculos 37

4.4.2 Ambiente com obstáculos 43

5 Conclusões e Trabalhos Futuros **49**

Referências Bibliográficas **51**

Lista de Figuras

2.1 Autômato determinístico G. 6
2.2 Mapa histórico da cidade de Königsberg, cujas 7 pontes estão assinaladas em vermelho. 8
2.3 Grafo G. 9
2.4 Resolução passo a passo do exemplo 4. 10

3.1 Respostas do sistema a degraus de velocidade de $0,1m/s$, $0,2m/s$ e $0,3m/s$. 17
3.2 Área A sobre a curva da resposta ao degrau de um sistema. 17
3.3 Comparação das respostas do sistema a degraus de velocidade linear com as respostas de G_L. 19
3.4 Comparação das respostas do sistema a degraus de velocidade angular com as respostas de G_R. 19
3.5 Diagrama de blocos para o sistema controlado. 20
3.6 Lugar das raízes do deslocamento linear em malha aberta. 21
3.7 Trajetória real do robô, tendo como referência um quadrado com arestas de $1m$. 23
3.8 Trajetória real do robô, tendo como referência uma estrela cuja distância entre os vértices é igual a $1m$. 23

4.1 Mapa do ambiente estruturado com suas dimensões. 26
4.2 Posição dos estados do autômato M no mapa. 26
4.3 Diagrama de transição de estados do autômato M. 28
4.4 Posição dos estados do autômato M no mapa, incluindo os estados do objetivo em vermelho. 31
4.5 Robô *Pioneer P3DX* controlado por um computador portátil. 36
4.6 Conteúdo da classe PID. 37
4.7 Conteúdo da classe *Pioneer P3DX*. 38
4.8 Conteúdo da classe **Matriz** 39
4.9 Estrutura da classe Autômato 40
4.10 Estrutura do Programa de execução 41

4.11 Ambiente estruturado com obstáculo. 42
4.12 Experimentos sem obstáculos. 42
4.13 Conteúdo da classe *Pioneer P3DX* modificada. 44
4.14 Estrutura da nova classe Autômato 45
4.15 Rotina executada após a detecção de um obstáculo. 46
4.16 Ambiente estruturado com obstáculo. 47

Lista de Tabelas

4.1 Possíveis movimentos do robô. 27
4.2 Tabela da função $f(X, E) \to X$ do autômato M. 29
4.3 Tabela de modificações na função $f : X \times E \to X$ do autômato M. . 31

Capítulo 1

Introdução

O emprego de robôs em ambientes industriais significou uma verdadeira revolução na capacidade produtiva humana, tanto em termos de qualidade quanto de quantidade; tornado-se um dos elementos mais impactantes da 3ª revolução industrial. Desta forma, a quantidade de robôs empregados na indústria vem crescendo com o passar dos anos [1]. Dentre os tipos de robôs que mais se popularizaram estão os veículos guiados automaticamente (VGA).

Os VGA são utilizados para transporte de carga desde a década de 1950, principalmente para materiais pesados ou perigosos. Com os avanços da tecnologia, porém, tais robôs tornaram-se mais versáteis, mais baratos e, consequentemente, mais numerosos [2]. Desta maneira, o interesse no desenvolvimento de formas mais eficientes de navegação em ambientes industriais vem crescendo.

Em ambientes industriais, geralmente conhecemos *a piori* a configuração de suas máquinas, esteiras, prateleiras e demais elementos, caracterizando-o como um ambiente estruturado, o que facilita a navegação dos VGA. Neste trabalho, consideramos o caso em que um robô trafega em um armazém industrial com a tarefa de transportar objetos de um ponto de carga e descarga até uma das prateleiras do armazém e vice-versa. Para tanto, o objetivo deste trabalho é desenvolver um sistema de planejamento de trajetória e navegação em um ambiente industrial estruturado com possíveis obstáculos desconhecidos *a priori*.

Este trabalho foi inspirado pela tese de mestrado de Lucas Molina [3], na qual é proposta um problema similar ao aqui apresentado para o mesmo robô *Pioneer P3DX*, bem como a proposta de modelagem do ambiente estruturado por um autômato. Foi inspirado ainda pelo Projeto de Graduação de Daniel Vaz [4], que propôs os métodos de modelagem e de controle para o modelo do simulador *Mobilesim* do robô *Pioneer P3DX*, aqui implementados na plataforma real. Quanto ao método de busca do menor caminho; ambos os trabalhos propuseram uma busca exaustiva, criando uma árvore de busca em que todas as possibilidades eram descobertas. Houve ainda inspiração da tese de mestrado de Gustavo Viana [5], na qual

é apresentada a álgebra *max-plus*, da qual deriva a álgebra *min-plus*.

Na abordagem que iremos propor, modelaremos o ambiente estruturado por meio de um autômato e, com base nesse modelo do ambiente, utilizaremos a álgebra *min-plus* matricial para implementar o algoritmo de Floyd-Warshall 2 com objetivo de calcular o menor caminho entre o ponto de carga e descarga e o ponto de destino, desviando de possíveis obstáculos desconhecidos *a priori*. Todo o sistema será desenvolvido em Python,utilizando a plataforma ROS e testado em uma plataforma real *Pioneer P3DX*.

Esse trabalho está organizado da seguinte forma. No capítulo 2, são apresentados os fundamentos teóricos necessários à execução do projeto. No capítulo 3, são abordados os procedimentos e técnicas de controle empregados na navegação do robô pelo ambiente estruturado. O capítulo 4 contém as descrições dos algoritmos que calculam a melhor rota de navegação do robô e da implementação do projeto no robô *Pioneer P3DX*. Finalmente, no capítulo 5, é apresentada a conclusão e os trabalhos futuros.

Capítulo 2

Fundamentos Teóricos

Neste capítulo, serão apresentados os fundamentos teóricos necessários à elaboração e compreensão deste Projeto de Graduação. Portanto, serão abordados os seguintes assuntos: sistemas a eventos discretos, controlador PID, álgebra *min-plus* e os algoritmos de busca do menor caminho.

2.1 Sistemas a Eventos Discretos

A busca por explicações para os fenômenos que ocorrem ao seu redor é algo natural ao ser humano. Assim, durante a História, incontáveis pessoas se dedicaram à observação e ao estudo dos mais variados eventos possíveis. Foi nessa busca por relações de causa e efeito que foram formulados modelos previsíveis que explicassem fenômenos de nossa realidade, que podem então ser chamados de sistemas.

Definição 1 (Sistema [6])
Um sistema é uma porção limitada da realidade que possui características que nos permitem descrevê-lo. Tais características, quando quantificadas, chamamos de variáveis de estado.

O conjunto de variáveis de estado é denominado estado do sistema. Tais variáveis podem ser contínuas ou discretas, e a evolução da dinâmica do sistema pode ser regida conforme o tempo ou por meio de acontecimentos específicos. Com base no tipo das variáveis de estado e na forma como se dá a evolução do sistema, podemos definir os Sistemas a Eventos Discretos da forma a seguir.

Definição 2 (Sistemas a Eventos Discretos (SED) [6])
Sistemas a Eventos Discretos são aqueles cuja evolução é regida pela ocorrência assíncrona da certas ações discretas e instantâneas, chamadas eventos.

O conjunto de estados de um SED, denominado espaço de estados, é discreto. Ao evoluir, o SED é levado de um estado para outro. Essa transição se dá quando

ocorre um evento. Note que as transições de estado não são funções do tempo, mas de eventos assíncronos.

2.1.1 Linguagem

Ao trabalharmos com sistemas dinâmicos, nosso maior interesse é registrar a evolução do sistema. No caso de SED, a evolução do sistema é descrita de forma adequada pela sequência de estados visitados e pela sequência de eventos, $e_1e_2e_3...e_n$, que provocaram tais transições. A sequência armazena então a ordem de ocorrência dos eventos, mas não o instante em que acontecem. No entanto, tal forma de representação de um SED é, frequentemente, suficiente pois normalmente não há interesse em saber quando os eventos ocorrem. Todas as sequências de eventos possíveis de serem geradas por um SED caracterizam a linguagem desse sistema. A seguir, o conceito de linguagem é definido formalmente.

Definição 3 (Linguagem)
Uma linguagem definida sobre um conjunto de eventos E (alfabeto) é um conjunto de sequências finitas (palavras), formadas por eventos pertencentes a E.

Em adição às palavras compostas por eventos pertencentes a E, consideramos também a palavra vazia, denotada por ε. A seguir, são apresentados alguns exemplos de linguagens definidas sobre um conjunto de eventos E.

Exemplo 1
Sendo $E = \{e_1, e_2, e_3\}$ o conjunto de eventos possíveis; podemos definir as seguintes linguagens:

$L_1 = \{\varepsilon, e_1, e_2, e_3, e_1e_2, e_1e_3, e_2e_3, e_1e_2e_3\}$;
$L_2 = \{\varepsilon, e_1, e_1e_1e_2, e_2e_2e_2e_1\}$;
L_3=$\{$*todas as palavras possíveis*$\}$;

Como as linguagens são conjuntos, nos quais os elementos são palavras, então todas as operações definidas para conjuntos também são aplicáveis às linguagens. Além disso, outras operações com palavras e linguagens são definidas; dentre elas a concatenação, fecho de Kleene e o fecho do prefixo.

Concatenação é a operação com a qual criamos palavras e, consequentemente, linguagens. A palavra *aba* é resultado da concatenação do evento a com a palavra ba, que por sua vez é a concatenação dos eventos b e a. Para duas linguagens L_1 e L_2 cujas palavras são compostas por eventos pertencentes a um dado E, a concatenação L_1L_2 será dada pela concatenação de cada uma das palavras de L_1 com todas as palavras de L_2 uma a uma.

Fecho de Kleene é a operação que permite formar uma linguagem infinita contável L^* contendo todas as palavras possíveis de serem formadas a partir das palavras

contidas numa certa linguagem L. Cabe notar que ao ser aplicada sobre E, gera uma linguagem E^* que contém todas as palavras arbitrariamente longas que podemos formar com os eventos de E, inclusive ε. Como exemplo, para a linguagem $L_0 = \{a, ab\}$, teremos $L_0^* = \{\varepsilon, a, ab, aa, aaa, aab, abaababa, \cdots\}$.

Seja $tuv = s$, com t, u, $v \in E^*$, então t é chamado de prefixo de s. O fecho do prefixo de um linguagem L, denotado por $\bar{L}$, é a operação que permite obter todos prefixos das palavras contidas na linguagem L, incluindo ε. Como exemplo, para a linguagem $L_1 = \{abcd\}$, teremos $\bar{L_1} = \{\varepsilon, a, ab, abc, abcd\}$.

A linguagem de um SED pode ser bastante complexa, mesmo sistemas simples podem gerar palavras arbitrariamente grandes e tornar difícil a representação de sua linguagem. Essa complexidade se propaga na execução de operação com essas linguagens, tornando necessário o uso de um modelo que permita executar operações de forma sistemática e eficiente. Um dos modelos mais difundidos são os autômatos, que será definido na próxima seção.

2.1.2 Autômatos

Um autômato é definido pela sêxtupla

$$G = (X, E, f, \Gamma, x_0, X_m),$$

em que:

- X é o conjunto de estados;
- E é o conjunto de eventos;
- $f : X \times E \to X$ é a função de transição de estados;
- $\Gamma \to 2^E$ é o conjunto de eventos ativos;
- $x_0 \in X$ é o estado inicial;
- $X_m \subseteq X$ é o conjunto de estados marcados.

Os autômatos também são conhecidos como máquinas de estado e podem ser representados por meio de um diagrama de transição de estados, formado por um grafo orientado cujos nós representam os estados e cujas arestas representam os eventos, como na figura 2.1. O estado inicial é indicado por uma seta e os estados marcados são indicados por círculos duplos.

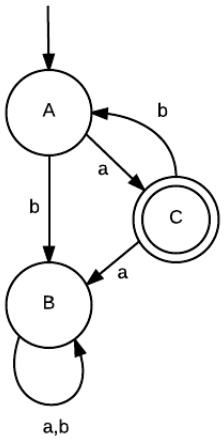


Figura 2.1: Autômato determinístico G.

Exemplo 2
Considere a seguinte sêxtupla $G = (X, E, f, \Gamma, x_0, X_m)$*, na qual teremos três estados,* $X = \{A, B, C\}$*, dos quais o estado inicial é* $x_0 = A$*, seus eventos são* $E = \{a, b\}$ *e temos um único estado marcado* $X_m = \{C\}$*. A função de transição de estados é dada por* $f(A, a) = C$*,* $f(A, b) = B$*,* $f(B, a) = B$*,* $f(B, b) = B$ *e* $f(C, a) = B$*. Note que o conjunto de eventos ativos para todos os estados é igual a* E*. O diagrama de transição de estados de* G *está representado na figura 2.1.*

A linguagem gerada por um autômato G, denotada por $L(G)$, é a linguagem que compreende todas as palavras que G pode gerar a partir do estado inicial. A linguagem marcada gerada por G, denotada por $L_m(G)$, contém apenas as palavras que alcança estados marcados a partir do estado inicial. Por exemplo, para o autômato G representado na figura 2.1, as linguagens gerada e marcada são dadas por $L(G) = \Sigma^*$ e $L_m(G) = \{(ab)^*a\}$, respectivamente.

2.2 Álgebra min-plus

A álgebra min-plus, também conhecida como álgebra tropical, é uma estrutura da forma $(\mathbb{R}_{\min}, \ominus, \otimes)$, na qual $\mathbb{R}_{\min}$ é igual ao conjunto dos números reais, $\mathbb{R}$, acrescido do elemento ∞ e as operações *min* $\ominus$ e *plus* $\otimes$ que são definidas da forma a seguir,

$$\begin{aligned} a \ominus b &= min(a, b), \\ a \otimes b &= a + b. \end{aligned}$$

Note que a operação *min* entre dois elementos a e b retorna o elemento de menor

valor. Por sua vez, a operação *plus* é semelhante à operação de soma. O elemento neutro da operação *min* é o ∞ e o elemento neutro da operação *plus* é o 0. Essas operações possuem as seguintes propriedades:

- Associatividade: $\forall a, b, c \in \mathbb{R}_{min}$; $a \ominus (b \ominus c) = b \ominus (a \ominus c)$; $a \otimes (b \otimes c) = b \otimes (a \otimes c)$
- Comutatividade: $\forall a, b \in \mathbb{R}_{min}$; $a \ominus b = b \ominus a$; $a \otimes b = b \otimes a$
- Distributividade: $\forall a, b, c \in \mathbb{R}_{min}$; $a \otimes (b \ominus c) = (a \otimes b) \ominus (a \otimes c)$
- Elemento Neutro: $\forall a \in \mathbb{R}_{min}$; $a \ominus \infty = \infty \ominus a = a$; $a \otimes 0 = 0 \otimes a = a$
- Absorção: $\forall a \in \mathbb{R}_{min}$; $a \otimes \infty = \infty \otimes a = \infty$
- Fechamento: Se $a > b \wedge a, b \in \mathbb{R}_{min}$; $a \ominus b = a \in \mathbb{R}_{min}$; $a \otimes b = c \in \mathbb{R}_{min}$

É possível estender as operações da álgebra $min - plus$ para matrizes. teremos outra definição das operações:

$$[A \ominus B]_{ij} = a_{ij} \ominus b_{ij} = min(a_{ij}, b_{ij}) \tag{2.1}$$

$$[A \otimes B]_{ik} = \ominus_{j=1}^{l} a_{ij} \otimes b_{ij} = min_{j \in l}\{a_{ij} + b_{jk}\} \tag{2.2}$$

Note que para a operação $min \ominus$, as dimensões das matrizes devem ser idênticas. Para a operação $plus \otimes$, o número de colunas de A precisa ser igual ao número de linhas de B.

Exemplo 3

Com o objetivo de ilustrar as operações min e plus matriciais, considere os seguintes exemplos:

Min:

$$\begin{bmatrix} 1 & -7 \\ 25 & 6 \end{bmatrix} \ominus \begin{bmatrix} 0 & 17 \\ 12 & 4 \end{bmatrix} = \begin{bmatrix} 1 \ominus 0 & -7 \ominus 17 \\ 25 \ominus 12 & 6 \ominus 4 \end{bmatrix} = \begin{bmatrix} 0 & -7 \\ 12 & 4 \end{bmatrix}$$

Plus:

$$\begin{bmatrix} 1 & -7 \\ 25 & 6 \end{bmatrix} \otimes \begin{bmatrix} 0 & 17 \\ 12 & 4 \end{bmatrix} = \begin{bmatrix} (1 \otimes 0) \ominus (-7 \otimes 12) & (1 \otimes 17) \ominus (-7 \otimes 4) \\ (25 \otimes 0) \ominus (6 \otimes 12) & (25 \otimes 17) \ominus (6 \otimes 4) \end{bmatrix}$$

$$= \begin{bmatrix} 1 & -3 \\ 18 & 10 \end{bmatrix}$$

2.3 Algoritmos de busca do Menor Caminho

A busca do menor caminho é importante na vida humana, pois faz parte de nossa natureza buscar o caminho de menor resistência para toda espécie de processo. A busca pelo melhor caminho se deu por processos heurísticos e empíricos até 1736, quando Leonhard Euler resolveu o problema das 7 pontes de Königsberg, representadas na figura 2.2 [7]. Com o questionamento se é possível cruzar as 7 pontes da cidade de Königsberg sem repetir alguma, deu-se início ao estudo da melhor trajetória. Mais tarde, no início do século XIX, Louis Poinsot e outros contemporâneos seus, dedicaram-se ao estudo de como desenhar figuras sem repetir vértices e sem levantar o lápis, dando início à Teoria de Grafos [8]. Não demorou muito tempo para que a conexão entre os dois tipos de problema fosse feita. No século XX, com o advento da computação, foram desenvolvidos diversos algoritmos para obter o menor caminho, dentre os quais podemos citar o Algoritmo de Dijkstra e o Algoritmo de Floyd-Warshall.

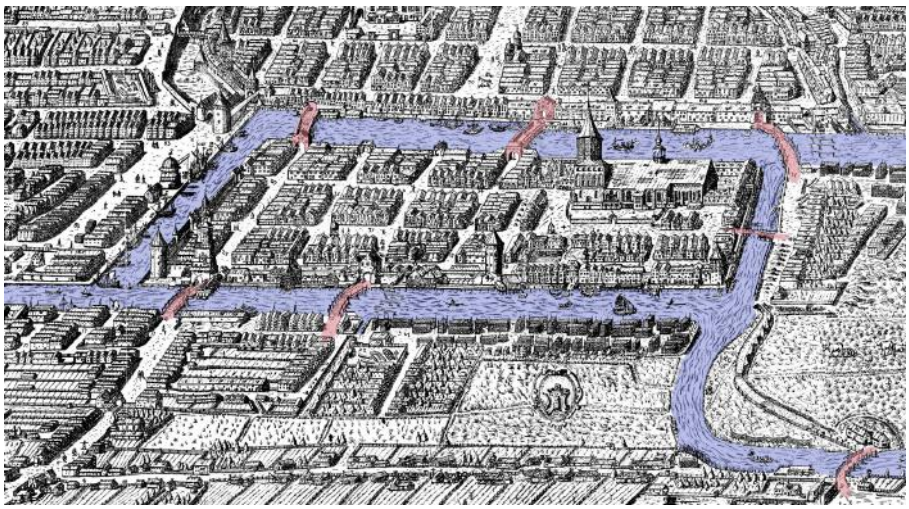


Figura 2.2: Mapa histórico da cidade de Königsberg, cujas 7 pontes estão assinaladas em vermelho.

2.3.1 Algoritmo de Dijkstra

O algoritmo de Dijkstra soluciona o problema do menor caminho para grafos, orientados ou não, ponderados com pesos não-negativos. Trata-se de um algoritmo guloso, porém, garante o melhor caminho de um vértice inicial ao final. Por ser de fácil implementação e de baixo custo computacional, é um dos algoritmos mais usados. Esse algoritmo sempre gera o menor caminho, pois parte do princípio de que a soma dos menores caminhos de um vértice ao outro dará o menor caminho total possível. Contudo, o algoritmo gera o menor caminho para um único par de origem e destino, não computando os demais pares [9].

A ideia do algoritmo de Dijkstra é basicamente encontrar a menor distância entre os vértices um a um. Nós iniciamos o algoritmo a partir do vértice inicial do caminho, percorremos as arestas que dele partem e assinalamos cada um dos vértices encontrados com o peso referente à aresta percorrida. Em seguida, nós escolhemos o vértice cuja distância assinalada foi a menor e, então, repetimos o mesmo processo para esse novo vértice. No entanto, para cada novo vértice visitado, o valor que atribuímos a ele é o peso da aresta que a ele leva somado ao valor atribuído ao vértice anterior do caminho. Caso um vértice já tenha sido visitado, nós verificamos se a nova distância é menor que a antiga. Caso seja, substituímos o valor atribuído ao vértice. Caso contrário, nada é feito. Devemos, então, repetir esse processo até que todos os vértices tenham sido visitados. Apresentaremos o exemplo 4 para consolidar o entendimento do algoritmo acima exposto.

Exemplo 4

Desejamos, a partir do vértice ***a*** *do grafo da figura 2.3, chegar ao vértice* ***e*** *percorrendo o menor caminho possível. Para tal, usaremos o algoritmo de Dijkstra.*

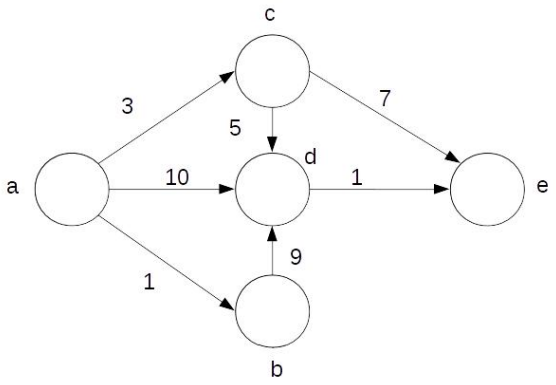
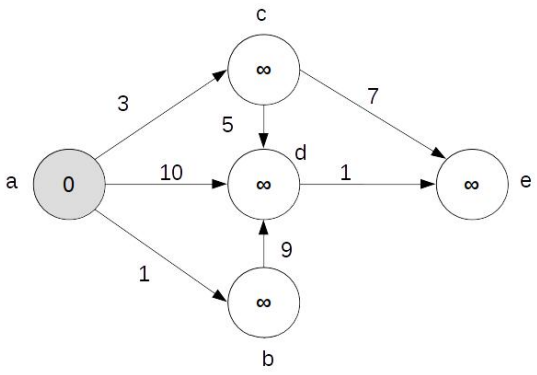
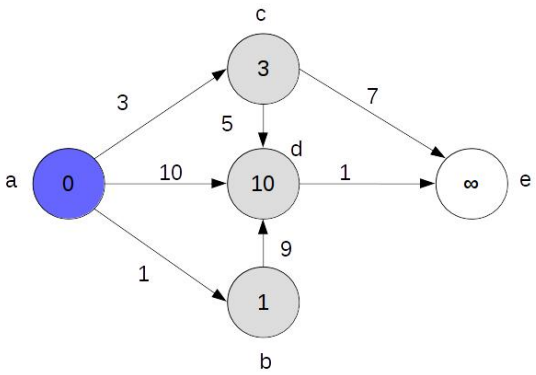


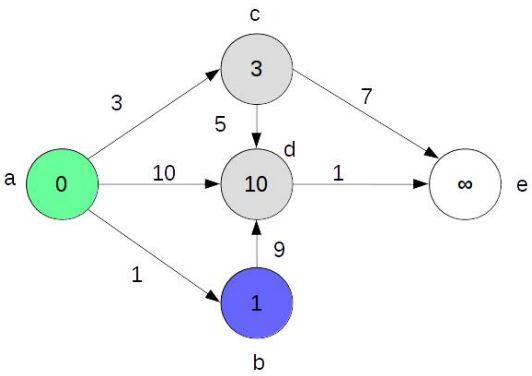
Figura 2.3: Grafo G.



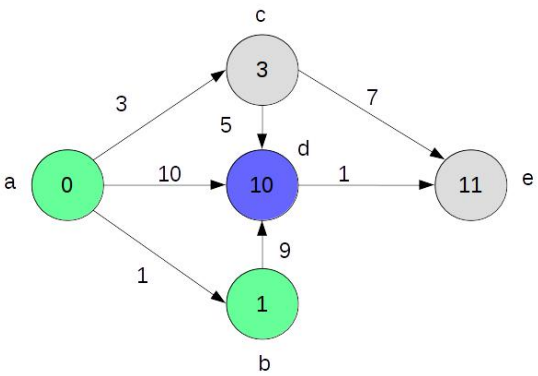
(a) Inicialização



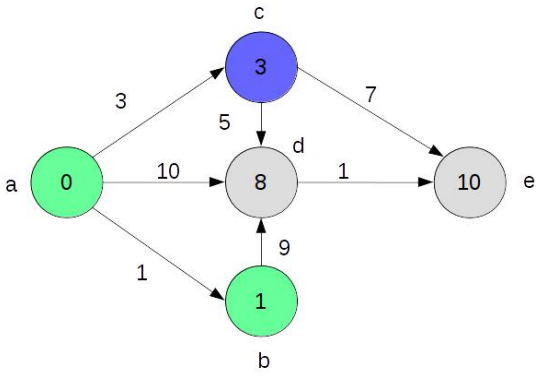
(b) 1º passo



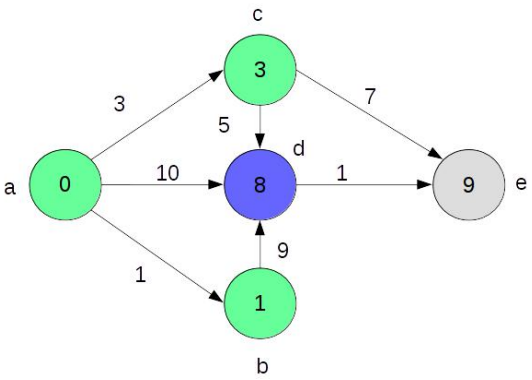
(c) 2º passo



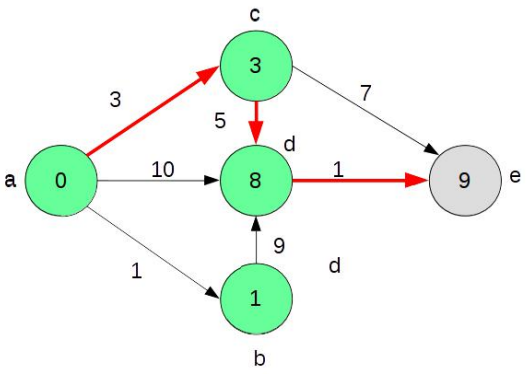
(d) 3º passo



(e) 4º passo



(f) 5º passo



(g) Menor caminho

Figura 2.4: Resolução passo a passo do exemplo 4.

Para inicializar o algoritmo, atribuímos a distância 0 ao vértice $\boldsymbol{a}$*, pois é nosso ponto de partida. Atribuímos, ainda, aos demais vértices a distância* ∞*, pois não foram alcançados por nenhuma aresta.*

Agora podemos começar as iterações. Para o 1º passo, partimos de $\boldsymbol{a}$*, percorremos as três arestas possíveis. Elas nos levam aos vértices* $\boldsymbol{b}$*,* $\boldsymbol{c}$ *e* $\boldsymbol{d}$*, aos quais serão atribuídas distâncias iguais aos pesos dos vértices que a eles levam somados à distância atribuída ao vértice* $\boldsymbol{a}$*. Desta forma, escolhemos o vértice* $\boldsymbol{b}$ *possuirá uma distância igual a* 1*,* $\boldsymbol{c}$ *igual a 3 e* $\boldsymbol{d}$ *igual a 10. É preciso escolher a partir de qual vértice continuaremos a iterar. Por ser um algoritmo guloso, devemos escolher aquele cuja aresta possua o menor peso. Desta forma, é vértice* $\boldsymbol{b}$*.*

O 2º passo segue os mesmos princípios do passo anterior. O Vértice $\boldsymbol{b}$ *possui uma única aresta de peso 9 que o liga ao vértice* $\boldsymbol{d}$*. Desconsideramos a aresta que o liga ao vértice* $\boldsymbol{a}$ *não só por conta do direcionamento, mas porque voltar a um vértice pelo qual já se passou somente aumentaria o caminho. Como o vértice* $\boldsymbol{d}$ *já possui uma distância atribuída igual a 10, precisamos verificar se a nova distância fornecida a partir do vértice* $\boldsymbol{b}$ *é menor. Assim, somamos a distância atribuída a* $\boldsymbol{b}$ *ao peso da aresta que liga a* $\boldsymbol{d}$*. Como o resultado é igual a 10, o peso é mantido. Seguimos então para o vértice* $\boldsymbol{d}$*.*

O 3º passo possui novamente uma única aresta viável que liga o vértice $\boldsymbol{d}$ *ao* $\boldsymbol{e}$*. Como essa aresta possui peso igual a 1 e a distância atribuída a* $\boldsymbol{d}$ *é igual a 10, ao vértice* $\boldsymbol{e}$ *será atribuída uma distância igual a 11.*

Como nem todos os vértices foram visitados, precisamos de mais iterações para garantir que a menor rota foi encontrada. Portanto, a 4ª iteração retorna ao vértice $\boldsymbol{a}$ *e segue pela aresta com o segundo menor peso, que liga o vértice* $\boldsymbol{a}$ *ao vértice* $\boldsymbol{c}$*. O vétice* $\boldsymbol{c}$ *possui duas arestas, uma com peso 7 que liga ao vértice* $\boldsymbol{e}$ *e outra que liga ao vértice* $\boldsymbol{d}$*, com peso* 5*. A nova distância para o vértice* $\boldsymbol{d}$ *é 8; enquanto a distância do vértice* $\boldsymbol{e}$ *é mantida em 10. Resta agora progredir pela aresta de menor peso que leva ao vértice* $\boldsymbol{d}$*.*

O 5º passo atribuirá ao vértice $\boldsymbol{e}$ *uma distância igual a 9, que é menor que a antiga. Como todos os vértices foram visitados, temos o menor caminho entre todos os vértices; o que nos permite, com uma sexta iteração, selecionar o menor caminho do vértice* $\boldsymbol{a}$ *ao vértice* $\boldsymbol{e}$*, percorrendo os vértices* $\boldsymbol{a}$*,* $\boldsymbol{c}$*,* $\boldsymbol{d}$ *e* $\boldsymbol{e}$*.*

O exemplo 4 mostra a eficácia do algoritmo de Dijkstra. Porém, executá-lo manualmente torna-se cada vez mais difícil quanto maior for o grafo. Assim, é importante criar um algoritmo para que o computador posso calcular o melhor caminho para grafos mais elaborados. O pseudocódigo para tal algoritmo é, portanto, apresentado abaixo.

Algoritmo 1: algoritmo de Dijkstra

```
Entrada: Grafo e Vértice de Origem
Saída: Distâncias entre os vértices para uma origem
início
    Q=[]
    para cada vértice v do Grafo faça
        dist[v] ← ∞ #distâncias desconhecidas
        ant[v] ← INDEFINIDO #nó anterior do caminho
        adicione v a Q #vértices não visitados
    fim
    dist[origem] ← 0
    enquanto Q for não-vazio faça
        u ← vértice em Q com menor dist[u] # origem primeiro
        remover u de Q
        para cada vértice vizinho v' de u faça
            alt ← dist[u] + peso(u, v)
            se alt < dist[v] então
                dist[v] ← alt
                ant[v] ← u
            fim
        fim
    fim
    retorna dist[],ant[]
fim
```

A versão mais simples do algoritmo de Dijkstra possui custo computacional de ordem $O(V^2)$, sendo V o número de vértices. Porém, se transformarmos a lista `Q` , após ser carregada com todos os vértices do grafo, em um *heap* binário[1],o custo do algoritmo cai para $O(A + V\,log(V))$, sendo A o número de arestas [9].

Vale ressaltar que a complexidade computacional apresentada é para apenas uma das origens possíveis. Se quisermos obter todas as menores distâncias entre todos os vértices, o algoritmo precisa ser executado V vezes, considerando todas as origens possíveis. Isso aumentaria o custo para $O(AV + V^2\,log(V))$.

2.3.2 Algoritmo de Floyd-Warshall

O algoritmo de Floyd-Warshall [9] resolve o problema do cálculo de todos os menores caminhos entre os vértices de um grafo ponderado. Ao contrário do Algoritmo de Dijkstra, ele funciona para grafos com arestas de pesos negativos.

[1]Um *heap* binário é uma estrutura de dados na qual uma lista é representada por uma árvore binária. O valor de um nó filho, para *heaps* máximos, não pode ser maior que o valor de seu nó pai; sendo o oposto para *heaps* mínimos [9].

O algoritmo de Floyd-Warshall precisa da matriz P, cujos elementos p_{ij} são iguais ao peso para ir do vértice j ao i, para um dado grafo G. Caso não haja conexão entre os vértices, o peso atribuído será ∞.

Este algoritmo é constituído por triplas, baseadas num vértice k, contendo todos os pares (i, j) de vértices pertencentes ao grafo G. Como os vértices serão rotulados de 1 a n, o índice do vértice k é igual ao valor do contador de iterações. Essas triplas serão examinadas por desigualdades da forma,

$$d_{ij}^{k} = min(d_{ij}^{k-1}, (d_{ik}^{k-1} + d_{kj}^{k-1})), \tag{2.3}$$

sendo d_{ij}^{k} um elemento da matriz D^k que contém a menor distância calculada para k iterações. O número k de iterações deve, ainda, ser igual ao número de vértices do grafo, para garantir que o menor caminho foi encontrado.

É possível apresentar, então, o seguinte pseudocódigo do algoritmo de Floyd-Warshall.

Algoritmo 2: algoritmo de Floyd-Warshall

Entrada: `Grafo` G `e a matriz de peso` P
Saída: `Matriz das distâncias` D
início
  $D^0 \leftarrow P$
  para $k = 1, 2, 3, \cdots, n$ **faça**
    se $d_{ik} + d_{jk} < d_{ij}$ **então**
      $d_{ij} \leftarrow d_{ik} + d_{kj}$
    fim
  fim
  retorna D
fim

O algoritmo apresentado possui complexidade computacional de ordem $O(V^3)$, sendo V o número de vértices do grafo.

Apresentaremos o exemplo 5 para consolidar o entendimento do algoritmo de Floyd-Warshall.

Exemplo 5

Para o grafo G da figura 2.3, encontre os menores caminhos entre os vértices usando o algoritmo de Floyd-Warshall.

Precisamos, primeiramente, indexar os vértices, pois seus nomes são letras e não

correspondem a posições na matriz.

$$\boldsymbol{a} : 1$$
$$\boldsymbol{b} : 2$$
$$\boldsymbol{c} : 3$$
$$\boldsymbol{d} : 4$$
$$\boldsymbol{e} : 5$$

Assim, podemos montar a matriz P observando os pesos das arestas de G.

$$P = \begin{bmatrix} 0 & \infty & \infty & \infty & \infty \\ 1 & 0 & \infty & \infty & \infty \\ 3 & \infty & 0 & \infty & \infty \\ 10 & 9 & 5 & 0 & \infty \\ \infty & \infty & 7 & 1 & 0 \end{bmatrix} = D^0$$

Note que os elementos p_{ij} da matriz P são iguais ao peso para ir do vértice j ao vértice i do grafo G e quando não existe conexão entre os vértices, o peso atribuído a p_{ij} será ∞, se $i \neq j$. Para o caso $i = j$, é definido que $p_{ij} = 0$.

A partir daí, podemos começar as interações do algoritmo, para $k = 1, 2, \cdots, 5$, pois o grafo possui 5 vértices. Lançaremos então mão da desigualdade triangular dada pela equação 2.3 para os elementos da matriz D^{k-1}. Quando caso $k = 1$, obtemos a seguinte matriz,

$$D^1 = \begin{bmatrix} 0 & \infty & \infty & \infty & \infty \\ 1 & 0 & \infty & \infty & \infty \\ 3 & \infty & 0 & \infty & \infty \\ 10 & 9 & 5 & 0 & \infty \\ \infty & \infty & 7 & 1 & 0 \end{bmatrix};$$

Note que o elemento $d^1_{2,1}$ é calculado por $d_{2,1} + d_{1,1}$, se $d_{2,1} + d_{1,1} < d_{2,1}$; ou seja, nesse caso o valor de $d_{2,1}$ é mantido para $d^1_{2,1}$.

Repetindo o procedimento do algoritmo até $k = 5$, teremos:

$$D^5 = \begin{bmatrix} 0 & \infty & \infty & \infty & \infty \\ 1 & 0 & \infty & \infty & \infty \\ 3 & \infty & 0 & \infty & \infty \\ 8 & 9 & 5 & 0 & \infty \\ 9 & 10 & 6 & 1 & 0 \end{bmatrix};$$

Teremos, assim, a distância entre todos os vértices do grafo. Não obtemos, contudo, a melhor rota, que deve ser obtida analisando a vizinhança de cada vértice e verificando, na linha correspondente ao destino, qual vértice que possui a menor distância, somada ao peso da aresta que liga os dois vértice , até chegarmos à coluna de destino.

No caso em que saímos do vértice ***a*** *para chegar ao vértice* ***e****, devemos analisar na 5ª linha da matriz as distâncias dos vizinhos de* ***a****, que são as colunas 2,3 e 4. Somando o peso das arestas que levam até eles, obtemos as distâncias 11, 9 e 11. Como 9 é o menor valor, passamos à coluna 3. Analisamos sua vizinhança, as colunas 4 e 5. Somando os pesos das arestas às distâncias da matriz, obtemos 6 e 7. Assim, passamos à coluna 4 e, como possui um único vizinho, à coluna 5, chegando finalmente ao destino. Portanto, o menor caminho passa pelas arestas* ***a****,****c****,****d*** *e* ***e****.*

Capítulo 3

Controle de Navegação

3.1 Introdução

O objetivo deste capítulo é explicar o desenvolvimento de um sistema de controle para navegação do robô em um ambiente estruturado com possíveis obstáculos. Para tanto, foi necessário levantar o modelo matemático do deslocamento do robô Pioneer P3DX. Como esse robô recebe apenas comandos de velocidade, mas possui odometria, tornou-se necessário modelar o sistema para entradas de velocidade e modelar seu odômetro, para que a saída do sistema seja a distância percorrida.

Como estamos interessados em deslocamentos tanto lineares quanto angulares, foi preciso particionar a modelagem em dois sub-sistemas, um para o movimento linear e outro para o angular. Contudo, para o movimento rotacional, verificou-se que um controlador proporcional bastava para controlar o giro do robô.

Com base nos modelos obtidos, lançando mão do método do lugar das raízes, foram desenvolvidos dois controladores em Python para a plataforma ROS. Essa plataforma permite trabalhar com código em tempo real e fornece diversas funções utilizadas no desenvolvimento do sistema de navegação do robô. O primeiro controlador atua sobre o deslocamento linear e o outro controla o movimento rotacional.

3.2 Modelagem

Para projetar o controle de navegação foi preciso levantar o modelo da Plataforma Pioneer P3DX. Primeiramente, analisamos a resposta da velocidade linear do robô a um degrau de velocidade. Foram aplicados degraus com amplitude $0,1m/s$, $0,2m/s$ e $0,3m/s$ e as respostas do sistema estão apresentadas na figura 3.1.

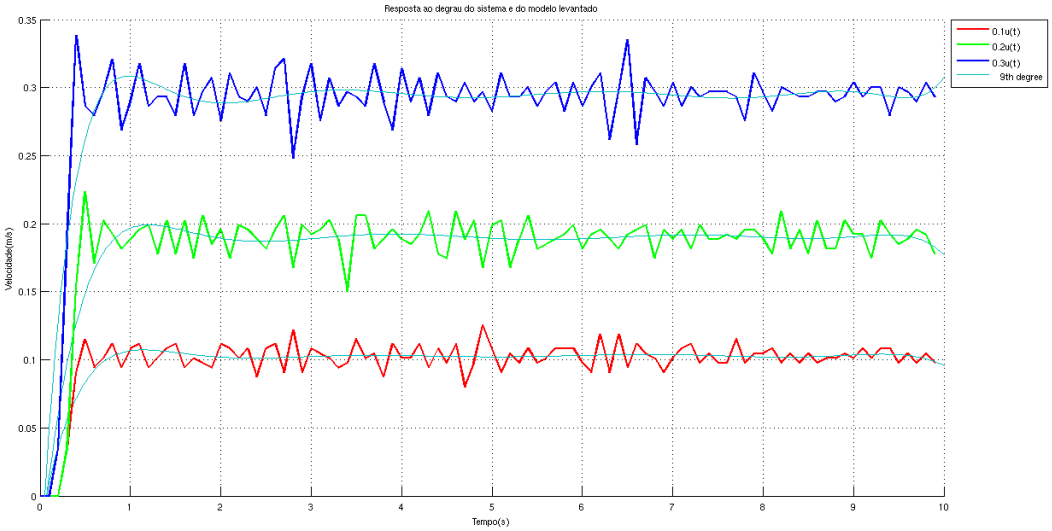


Figura 3.1: Respostas do sistema a degraus de velocidade de $0,1m/s$, $0,2m/s$ e $0,3m/s$.

Como podemos observar na figura 3.1, o sistema pode ser aproximado por uma função de transferência de 2ª ordem criticamente amortecida, da seguinte forma,

$$G_L(s) = \frac{V_s(s)}{V_e(s)} = \frac{K}{(\tau s + 1)^2}. \tag{3.1}$$

Precisamos, agora, desenvolver um método para obter os parâmetros do sistema, K e τ. Escolheu-se, para tal, o método da área [10] para sistemas de 2ª ordem, o qual será abordado a seguir.

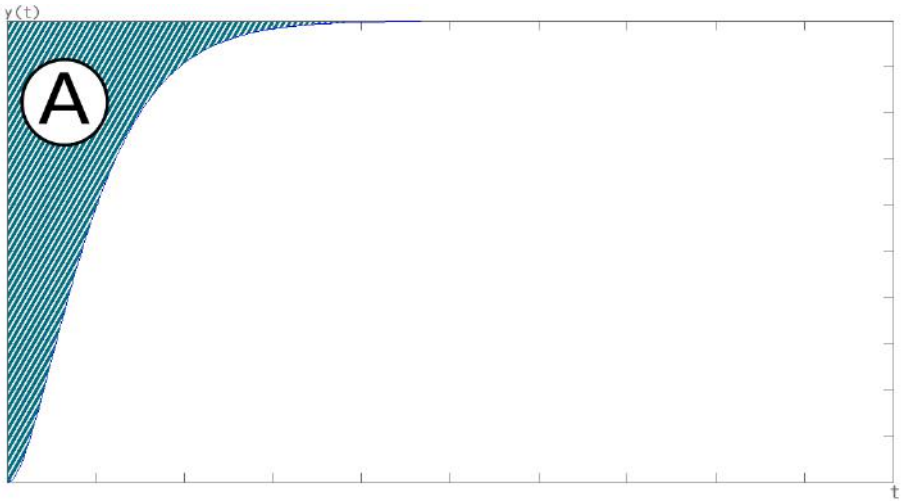


Figura 3.2: Área A sobre a curva da resposta ao degrau de um sistema.

Para um degrau de amplitude B, considerando um sistema como o da equação 3.1, temos que a área sobre a curva da resposta ao degrau, como indicada

na figura 3.2, é dada por,

$$A = KB \int_0^\infty (e^{\frac{-1}{\tau}t} + \frac{1}{\tau} t e^{\frac{-1}{\tau}t}) dt \tag{3.2}$$

Portanto,

$$A = KB\tau - KB \int_0^\infty -\frac{1}{\tau} t e^{\frac{-1}{\tau}t} dt = KB\tau - KB \left[t e^{-\frac{1}{\tau}t} |_0^\infty \right] - KB \left[\int_0^\infty e^{\frac{-1}{\tau}t} dt \right] = 2KB\tau$$

Assim sendo,

$$\tau = \frac{A}{2KB}, \tag{3.3}$$

na qual K é obtido observando o valor da saída em regime estacionário, y_{EE}, em relação à referência B, como na equação 3.4. Porém, como o sinal odométrico apresentava bastante variação, foi necessário interpolar cada curva de respostas ao degrau do robô por um polinômio do 9º grau para se obter os valores necessários ao cálculo do ganho K.

$$K = \frac{y_{ee}}{B} \tag{3.4}$$

Para cada uma das curvas anteriormente obtidas foi possível calcular sua respectiva área A_i, com o auxílio de um programa de computação; bem como os valores de K_i. Desta maneira, pudemos obter os valores de τ_i e K_i para cada curva. Assim, a partir da média dos valores de τ_i e K_i, foram obtidos os valores de K e τ do sistema. Os valores calculados foram, então, $K = 0,99$ e $\tau = 0,37$, o que nos deixa com a seguinte função de transferência para $G_L(s)$,

$$G_L(s) = \frac{0,99}{(0,37s + 1)^2}. \tag{3.5}$$

Porém, como se deseja controlar a posição do robô, estamos interessados na posição odométrica do robô para o cálculo do erro. Foi preciso, então, modelar a dinâmica deste sensor. Analisando a posição do robô para os mesmos sinais de entrada, verificou-se que o sensor poderia ser aproximado por um integrador de ganho unitário.

O mesmo procedimento acima descrito foi repetido para o deslocamento angular, o que nos permitiu encontrar a seguinte função de transferência G_R,

$$G_R(s) = \frac{0,99}{(0,32s + 1)^2}. \tag{3.6}$$

Os modelos obtidos puderam ser validados por meio da comparação dos dados

obtidos do robô real com os resultados de simulações, utilizando as funções de transferência G_L e G_R. As comparações podem ser vistas nas figuras 3.3 e 3.4; para o modelo linear e para o modelo rotacional, respectivamente.

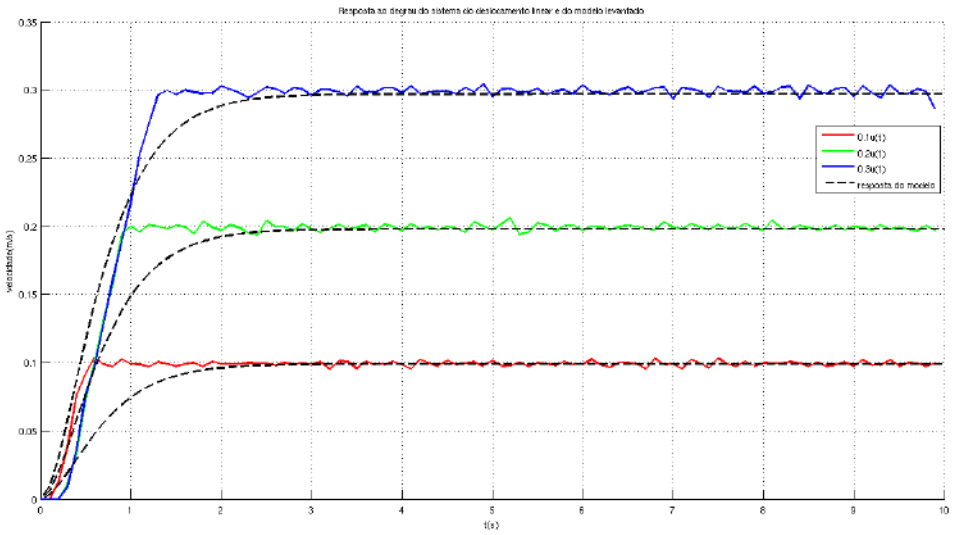


Figura 3.3: Comparação das respostas do sistema a degraus de velocidade linear com as respostas de G_L.

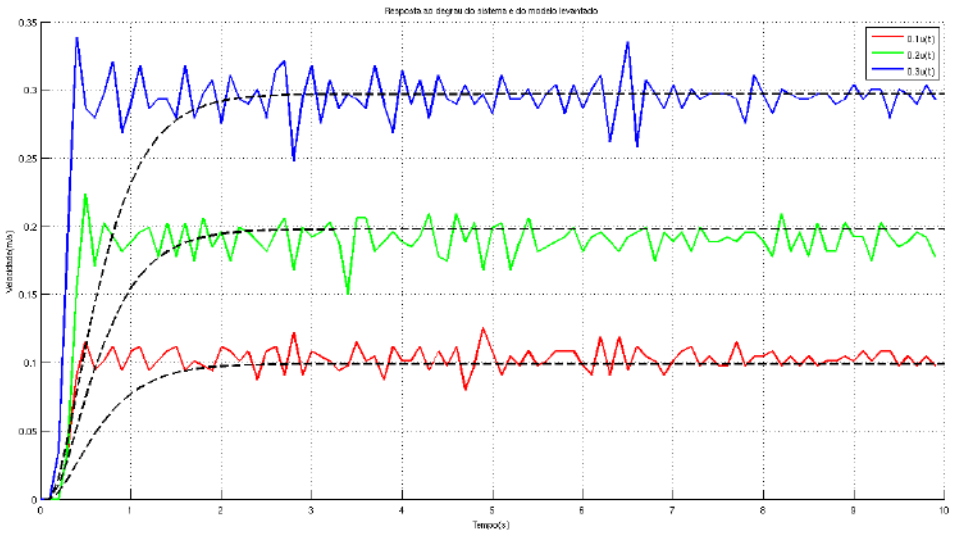


Figura 3.4: Comparação das respostas do sistema a degraus de velocidade angular com as respostas de G_R.

É possível observar que, apesar do tempo de subida superior, os modelos obtidos aproximam satisfatoriamente o comportamento do robô Pioneer P3DX, pois são realizados somente movimentos de translação e rotação desacoplados e que não necessitam de um modelo mais preciso. Com as funções de transferência do sistema

devidamente calculadas, foi possível calcular os controladores do deslocamento linear e do deslocamento angular.

3.3 Projeto do Controlador

O diagrama de blocos do sistema, com controlador, pode ser representado pela figura 3.5.

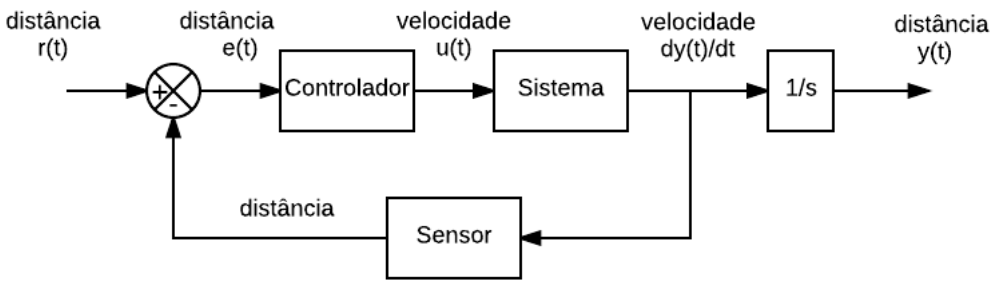


Figura 3.5: Diagrama de blocos para o sistema controlado.

A partir da equação (3.5), considerando a função de transferência do sensor, temos a seguinte função de transferência em malha aberta para o deslocamento linear,

$$G_T(s) = \frac{0,99}{s(0,37s+1)^2}. \tag{3.7}$$

Decidimos, a fim de escolher a melhor estrutura de controle, analisar o lugar das raízes do sistema dado pela equação 3.7.

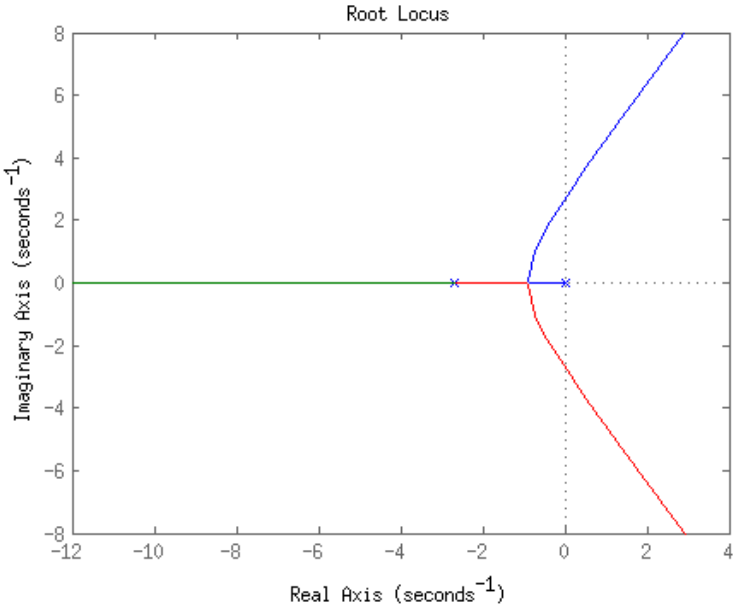


Figura 3.6: Lugar das raízes do deslocamento linear em malha aberta.

Por meio da figura 3.6, é possível ver que podemos melhorar o desempenho ao alocarmos um zero no semiplano esquerdo, de forma a cancelar um dos pólos. Desta maneira, o esquema de controle adotado foi o Proporcional e Derivativo (PD), cuja função de transferência é dada por,

$$G_{PD}(s) = \frac{K_P(s + \frac{1}{T_D})}{\frac{T_D}{N}(s + \frac{N}{T_D})} \tag{3.8}$$

Para cancelar um dos polos, alocamos o zero em $-\frac{1}{\tau}$. Desta forma, temos que $T_D = \tau = 0,37$. Por sua vez, N precisa ser grande com relação a τ, portanto, escolheu-se $N = 100$. Deseja-se que o sistema apresente um comportamento criticamente amortecido, portanto, K_P deve ser tal que leve os polos do sistema realimentado ao ponto de ramificação no lugar das raízes. Podemos calcular este ganho, pois sabemos que a derivada com relação a s do denominador da função de transferência do sistema $G_{LT}(s) = G_T(s)G_{PD}(s)$, $1 + GLT(s)$, é nula no ponto de ramificação. Desta forma, temos que:

$$1 + G_{LT}(s) = 1 + \frac{K_P(s + \frac{1}{0,37})}{\frac{0,37}{100}(s + \frac{100}{0,37})} \frac{0,99}{s(0,37s + 1)^2} = 0 \tag{3.9}$$

$$\frac{K_P(s + \frac{1}{0,37})}{\frac{0,37}{100}(s + \frac{100}{0,37})} \frac{0,99}{s[(s + \frac{1}{0,37})0,37]^2} = -1 \tag{3.10}$$

Simplificando a equação 3.10, obtemos:

$$\frac{K_P}{(s+270,27)}\frac{1954,1}{s(s+2,703)} = -1 \tag{3.11}$$

De onde obtemos:

$$K_P = -(s^3 + 272,97s^2 + 730,46s) \tag{3.12}$$

É preciso encontrar o ponto onde a derivada é nula; desta maneira, fazemos:

$$\frac{dK_P}{ds} = -\frac{d(s^3 + 272,97s^2 + 730,46s)}{ds} = 0 \tag{3.13}$$

De onde obtivemos a seguinte equação:

$$3s^2 + 545.94s + 730,46 = 0 \tag{3.14}$$

Resolvendo o sistema, obtivemos os seguintes valores possíveis para s

$$\begin{cases} s = -1,35 \\ s = -180,63 \end{cases}$$

Escolhemos $s = -1,35$, pois, de acordo com o lugar das raízes apresentado na figura 3.6, o ponto de ramificação está contido no intervalo $(0,2)$.

Podemos obter o valor de K_P substituindo o valor calculado de s na equação (3.12). Obtivemos então:

$$G^L_{PD}(s) = \frac{0,498(s + \frac{1}{0,37})}{0,0037s + 1} \tag{3.15}$$

A função de transferência do controlador do movimento angular foi obtida pelo mesmo método exposto acima, considerando a função de transferência apresentada na equação (3.6). Obteve-se, portanto, a função de transferência da equação (3.16) para o controle do movimento angular:

$$G^R_{PD} = \frac{0,015(s + \frac{1}{0,32})}{0,0032s + 1} \tag{3.16}$$

O controlador do deslocamento angular, contudo, recebeu uma ligeira modificação quanto ao cálculo do sinal de erro, conformando-o ao intervalo [-180,180], pois é o mesmo intervalo de valores que a orientação do robô pode assumir.

Com os controladores devidamente sintetizados, foram realizados dois testes para validar a eficácia dos mesmos. O primeiro teste consistiu em passar como trajetória ao robô um quadrado com aresta de comprimento igual a $1m$. O segundo teste consistiu em passar como trajetória ao robô uma estrela, cuja distância entres os vértices é igual a $1m$. Os resultados do primeiro e do segundo teste podem ser

observados, respectivamente, nas figuras 3.7 e 3.8.

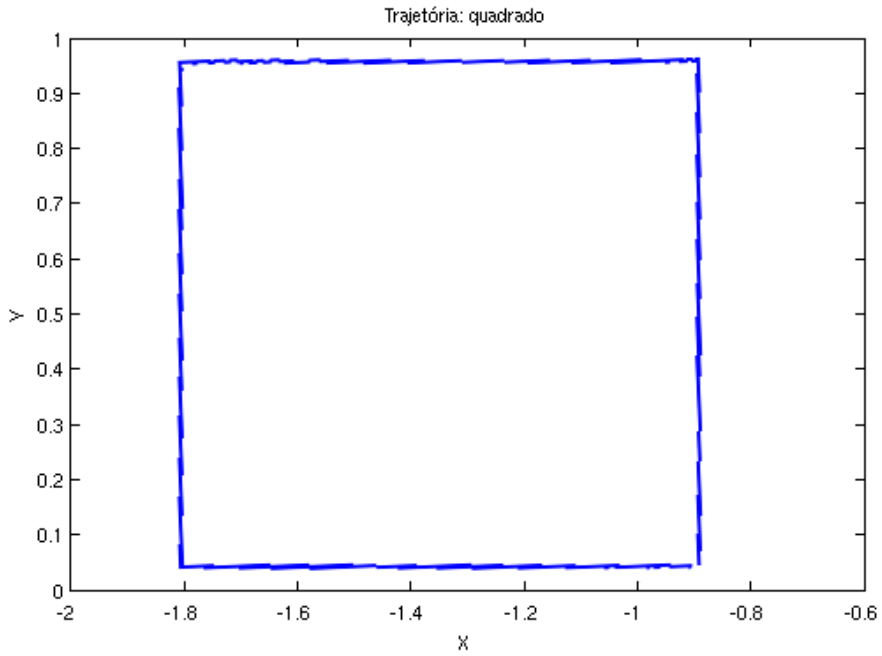


Figura 3.7: Trajetória real do robô, tendo como referência um quadrado com arestas de $1m$.

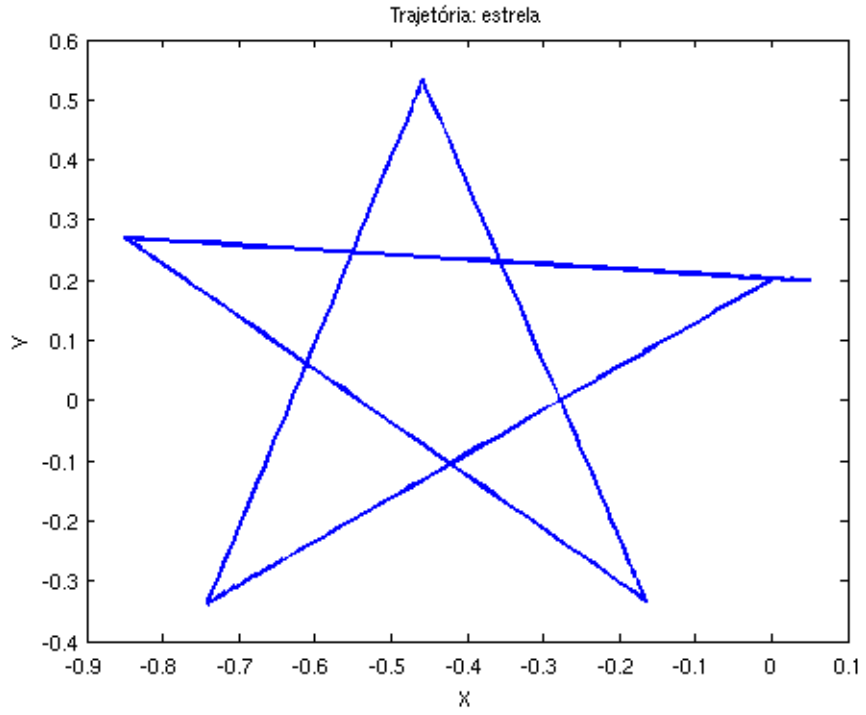


Figura 3.8: Trajetória real do robô, tendo como referência uma estrela cuja distância entre os vértices é igual a $1m$.

Por meio das figuras 3.7 e 3.8 verificamos que há um erro ao final de ambas as trajetórias, mais evidente na trajetória em forma de estrela do que na trajetória em

forma de quadrado. Esse erro deve-se, em parte, ao erro acumulado na odometria do robô ao longo do tempo. Contudo, como os movimentos a serem desempenhados no cumprimento da tarefa do robô são translações e rotações desacoplados, o desempenho dos controladores é suficiente para este trabalho.

Capítulo 4

Planejamento de Trajetória

4.1 Introdução

O objetivo deste capítulo é explicar como se deu o desenvolvimento do planejamento de trajetória para um dado ambiente estruturado. Primeiramente, na seção 4.2, é apresentado o mapa do ambiente estruturado pelo qual o robô deve navegar e sua modelagem por um autômato, conforme o formalismo apresentado na subseção 2.1.2 . A seguir, na seção 4.3, são apresentados os dois algoritmos que calculam a melhor rota para chegar ao destino, o algoritmo de Floyd-Warshall Min-Plus e o algoritmo de roteamento. Finalmente, na seção 4.4, é explicado como a modelagem realizada, os algoritmos de Floyd-Warshal min-plus e de roteamento e o sistema de controle foram implementados na prática, para ambientes sem obstáculos 4.4.1 e para ambientes com obstáculos 4.4.2.

4.2 Modelagem do ambiente utilizando autômato

Levando em consideração o contexto atual da utilização comercial de robôs, escolhemos para este trabalho o mapa da figura 4.1 como ambiente estruturado a ser estudado. Esse mapa pode representar um armazém, em que os retângulos cinza representam as prateleiras que armazenam itens diversos. A área no lado superior esquerdo do mapa representa um local de carga/descarga de materiais ou ainda um ponto de recarga do robô, onde esse deve ficar enquanto não estiver em uso. A tarefa do robô, é, portanto, estocar e retirar itens do armazém. Nesse trabalho, consideramos somente a tarefa de sair do ponto de recarga, chegar até um determinado ponto e retornar ao ponto de recarga. Não faz parte do escopo desse trabalho a manipulação do objeto retirado ou colocado nas prateleiras.

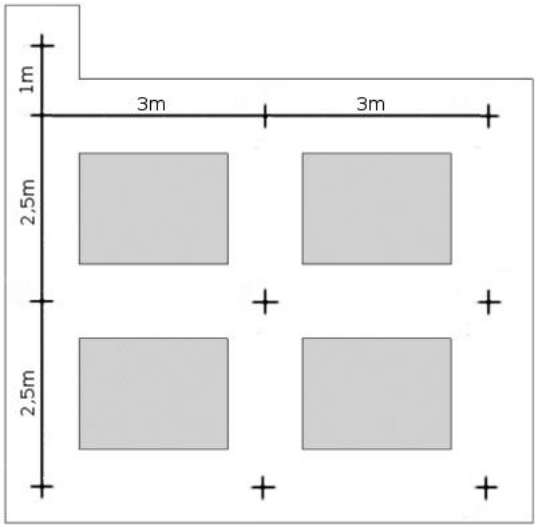


Figura 4.1: Mapa do ambiente estruturado com suas dimensões.

Para que o robô pudesse navegar no mapa, decidimos representar o armazém por um autômato M com 38 estados. Cada estado representa uma posição e uma orientação nas interseções dos corredores do mapa. Os eventos, por sua vez, representam a movimentação do robô de uma posição ou orientação para outra; conforme consta na tabela 4.1. O diagrama de transição de estados do autômato M é apresentado na figura 4.3.

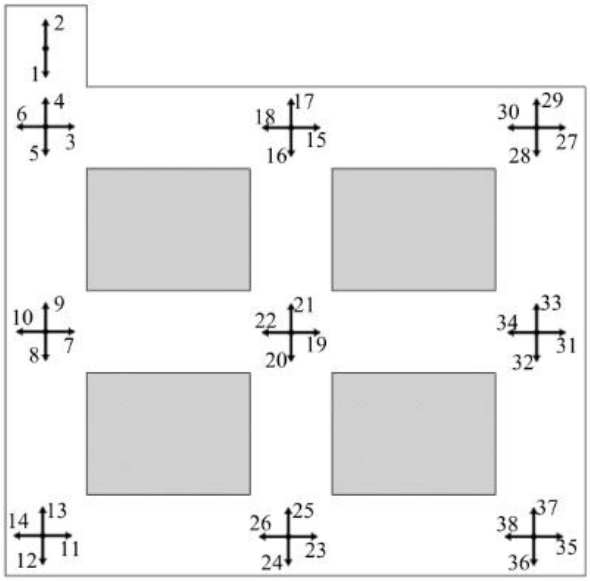


Figura 4.2: Posição dos estados do autômato M no mapa.

Tabela 4.1: Possíveis movimentos do robô.

Evento	Movimento do robô
$a1.0$	Deslocamento de $1m$
$a2.5$	Deslocamento de $2.5m$
$a3.0$	Deslocamento de $3m$
$a3.5$	Deslocamento de $3.5m$
$a5.0$	Deslocamento de $5m$
$a6.0$	Deslocamento de $6m$
$g90$	Giro de 90^o
$g-90$	Giro de -90^o
$g180$	Giro de 180^o

Formalmente, o autômato M é definido pela sêxtupla $M = (X, E, f, \Gamma, x_0, X_m)$, na qual:

- $X = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38\}$;

- $E = \{a1.0, a2.5, a3.0, a3.5, a5.0, a6.0, an, g90, g-90, g180\}$;

- $f : X \times E \to X$ é definida de acordo com a tabela 4.2;
- $\Gamma(x) = \{e \in E : f(x, e) \text{ é definida}\}$;
- $x_0 = 1$;
- $X_m = \{\}$.

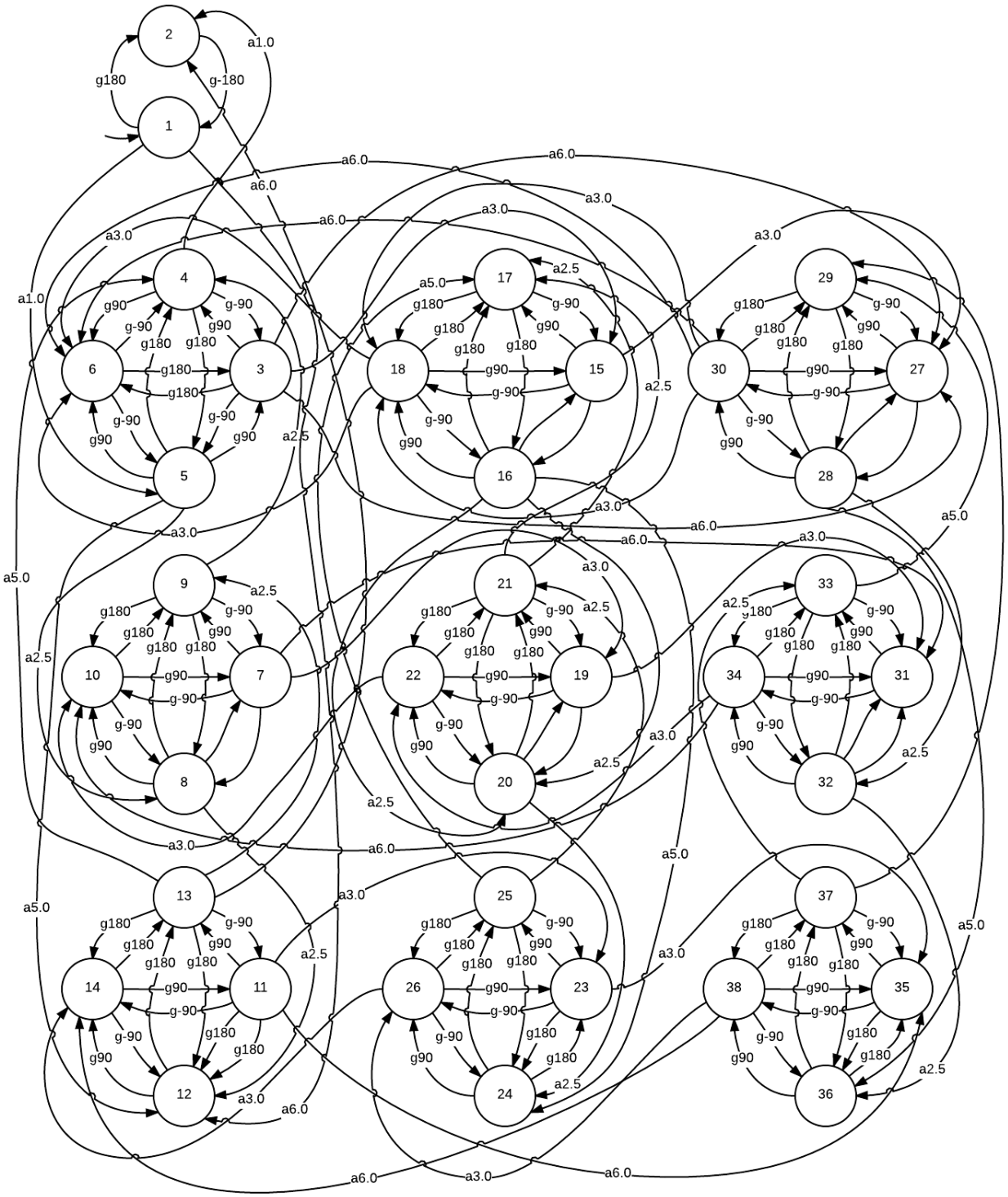


Figura 4.3: Diagrama de transição de estados do autômato M.

Tabela 4.2: Tabela da função $f(X, E) \rightarrow X$ do autômato M.

Estado atual	Evento	Próximo Estado
1	g180	2
	a1.0	5
	a3.5	8
	a6.0	12
2	g180	1
3	g90	4
	g-90	5
	g180	6
	a3.0	15
	a6.0	12
4	g90	6
	g-90	3
	g180	5
	a1.0	2
5	g90	3
	g-90	6
	g180	4
	a2.5	8
	a5.0	12
6	g90	5
	g-90	4
	g180	3
7	g90	9
	g-90	8
	g180	10
	a3.0	19
	a6.0	31
8	g90	7
	g-90	10
	g180	9
	a2.5	12
9	g90	10
	g-90	7
	g180	8
	a2.5	4
	a5.0	2
10	g90	8
	g-90	9
	g180	7
11	g90	13
	g-90	12
	g180	14
	a3.0	23
	a6.0	35
12	g90	11
	g-90	14
	g180	13
13	g90	14
	g-90	11
	g180	12
	a2.5	9
	a5.0	4
	a6.0	2
14	g90	12
	g-90	13
	g180	11
15	g90	17
	g-90	16
	g180	18
	a3.0	27
16	g90	15
	g-90	18
	g180	17
	a2.5	20
	a5.0	24
17	g90	18
	g-90	15
	g180	16
18	g90	16
	g-90	17
	g180	15
	a3.0	6
19	g90	21
	g-90	20
	g180	22
	a3.0	31

Estado atual	Evento	Próximo Estado
20	g90	19
	g-90	22
	g180	21
	a2.5	24
21	g90	22
	g-90	19
	g180	20
	a2.5	17
22	g90	20
	g-90	21
	g180	19
	a3.0	10
23	g90	25
	g-90	24
	g180	26
	a3.0	35
24	g90	23
	g-90	26
	g180	25
25	g90	26
	g-90	23
	g180	24
	a2.5	21
	a5.0	17
26	g90	24
	g-90	25
	g180	23
	a3.0	14
27	g90	29
	g-90	28
	g180	30
28	g90	27
	g-90	28
	g180	29
	a2.5	32
	a5.0	36
29	g90	30
	g-90	27
	g180	28

Estado atual	Evento	Próximo Estado
30	g90	28
	g-90	29
	g180	27
	a3.0	18
	a6.0	6
31	g90	33
	g-90	32
	g180	34
32	g90	31
	g-90	34
	g180	33
	a2.5	36
33	g90	34
	g-90	31
	g180	32
	a2.5	29
34	g90	32
	g-90	33
	g180	31
	a3.0	22
	a6.0	10
35	g90	37
	g-90	36
	g180	38
36	g90	35
	g-90	38
	g180	37
37	g90	38
	g-90	35
	g180	36
	a2.5	33
	a5.0	29
38	g90	36
	g-90	37
	g180	35
	a3.0	26
	a6.0	14

O modelo matemático para o mapa definido pelo autômato M representa bem as interseções dos corredores, porém, não permite ao robô chegar a pontos intermediários para estocar ou retirar itens das prateleiras. Desta forma, observou-se a necessidade de criar dois estados marcados, 39 e 40, para representar o objetivo final do robô no mapa. A posição desses estados é definida pelo usuário no início de cada

tarefa, por coordenadas $(x; y)$ em relação ao sistema de coordenada E_0, mostrado em azul na figura 4.4. Um exemplo pode ser visto na figura 4.4, onde os estados 39 e 40 foram alocados nas coordenadas $(4; 2, 5)$.

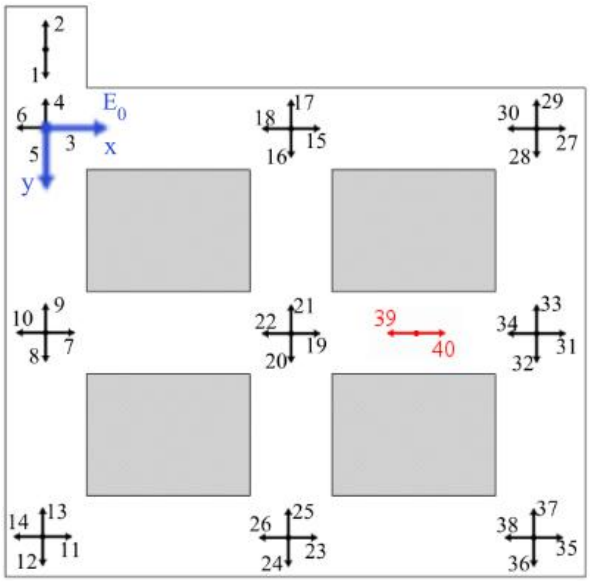


Figura 4.4: Posição dos estados do autômato M no mapa, incluindo os estados do objetivo em vermelho.

Os eventos que levam aos estados 39 e 40 são definidos pela própria função que acrescenta esses estados ao autômato M, de acordo com a distância aos estados mais próximos. No caso ilustrado na figura 4.4, $f(e, x)$ recebe as modificações apresentadas na tabela 4.3.

Tabela 4.3: Tabela de modificações na função $f : X \times E \to X$ do autômato M.

Estado atual	Evento	Próximo Estado
19	g90	21
	g-90	20
	g180	22
	a1.0	40
	a3.0	31
34	g90	32
	g-90	33
	g180	31
	a2.0	39
	a3.0	22
	a6.0	10
39	g180	40
	a1.0	22
40	g180	39
	a2.0	31

O autômato M, depois de sofrer essas alterações, passa a ser chamado de M'. Com a definição do mapa e dos objetivos da tarefa por meio do autômato M', o passo seguinte é implementar um algoritmo que determine quais eventos compõem o menor caminho entre a origem e o destino.

4.3 Algoritmos de planejamento de trajetória

A seção 2.3.2 apresenta uma das formas de implementação do algoritmo de Floyd-Warshall. Neste trabalho, desenvolvemos uma versão que substitui as desigualdades triangulares por operações da álgebra $min - plus$ com matrizes. Essa versão chamaremos, portanto, de algoritmo de Floyd-Warshall *min-plus*.

O algoritmo de Floyd-Warshall *min-plus* é estruturado a partir da matriz de pesos P, obtida diretamente do autômato M que representa o mapa. Para criá-la, faz-se com que o elemento $p_{j,k}$ de P seja o peso do evento que leva o autômato do estado j ao k. Assim, cada linha contém o peso para ir do estado j a todos os demais estados. Caso não haja conexão entre os estados, o peso deve ser ∞.

Cada iteração i do algoritmo corresponde a uma transição do autômato, ou seja, ao $i+1$-ésimo passo dado pelo robô para chegar ao destino. Chamaremos, portanto, as iterações de passos. A cada um desses passos, calcularemos uma matriz contendo a distância entre os vértices j e k para $i+1$ passos, chamada P_i. Como o pior caminho possível para um autômato de n estados possui $n-1$ passos, o algoritmo precisará de $n-2$ iterações, pois P representa o primeiro passo.

Precisamos, ainda, criar uma matriz D para armazenar as menores distâncias entre os vértices a cada iteração do algoritmo de Floyd-Warshall *min-plus*. Assim, por meio da operação $D \ominus P_i$, garantimos que teremos o menor caminho em D. Cabe ressaltar que, na inicialização do algoritmo, a matriz D será igual a P.

Para executar o algoritmo de Floyd-Warshall *min-plus* precisamos utilizar duas operações. A primeira é a operação *plus* matricial, definida em (2.2), entre as matrizes P e P_{i-1}, e a segunda é a operação min matricial, definida em (2.1) entre as matrizes P_i e D. O procedimento detalhado do algoritmo de Floyd-Warshall $min - plus$ é apresentado a seguir.

Algoritmo 3: algoritmo de Floyd-Warshall *min-plus*

Entrada: `Matriz de pesos P e nº de estados n`

Saída: `Matriz das distâncias entre os vértices para n-1 passos`

início

```
i = 1
P_{i-1} = P
D = P
se n> 2 então
    enquanto i < n − 1 faça
        P_i = P ⊗ P_{i-1}
        D = D ⊖ P_i
        i = i + 1
    fim
fim
retorna D
```

fim

A cada iteração do algoritmo, é realizada uma operação de *plus* e uma operação de *min*. A operação de *plus* possui um custo computacional igual ao de uma multiplicação entre duas matrizes, de ordem $O(V^3)$, sendo V o número de vértices do grafo. Já a operação *min* é da ordem de $O(V^2)$, igual à de uma soma de matrizes. Como essas operações serão executadas $V - 1$ vezes, o peso total do algoritmo de Floyd-Warshall *min-plus* é igual a $(V - 1)(V^3 + V^2) = V^4 - V^2$, ou seja, possui complexidade computacional de pior caso de ordem $O(V^4)$.

Contudo, o algoritmo de Floyd-Warshall *min-plus* sozinho não nos permite encontrar a melhor rota. É preciso lançar mão de um algoritmo de roteamento. Esse algoritmo, tem como entrada a matriz D, que contém as menores distâncias entre os estados. Ela será transposta a fim de facilitar a implementação do algoritmo, tornando-se D^T. Agora, o estado de origem passa a ser j e o estado de destino i, para as distâncias d_{ij}^T. Assim, precisamos iterar em uma linha apenas, pois estamos interessados nas distâncias de todos os estados ao estado final. Conhecendo o estado inicial, analisamos seus vizinhos e buscamos aquele que possui a menor distância até o estado final e_f. Repetimos o mesmo procedimento até chegarmos em e_f. O procedimento detalhado do algoritmo de roteamento é apresentado a seguir.

Algoritmo 4: algoritmo de roteamento

Entrada: `Matriz de Distâncias` D, `estado inicial` e_i `e estado de destino` e_f

Saída: `Vetor com o caminho` R

início

$D = D^T$

$e_a = e_i$ `#estado anterior`

R `= []`

enquanto $e_a \neq e_f$ **faça**

`distancias = []`

para *todos os vizinhos* e_i *de* e_a **faça**

`distancias` $\leftarrow$ `(`e_i`,peso(`e_i, e_f`)+ peso(`e_i, e_a`))`

fim

e_a `= min(distancias)`

$R \leftarrow e_a$

fim

retorna R

fim

No exemplo 6, ilustramos a aplicação do algoritmo de Floyd-Warshall *min-plus* .

Exemplo 6

Usando como base o grafo G da figura 2.3 e a matriz de pesos dele obtida no exemplo 5, precisamos obter o menor caminho entre os vértices ***a*** *e* ***e*** *usando o algoritmo de Floyd-Warshall min-plus.*

Para inicializar o algoritmo, ou seja, no 1º passo teremos:

$$P = \begin{bmatrix} 0 & 1 & 3 & 10 & \infty \\ \infty & 0 & \infty & 9 & \infty \\ \infty & \infty & 0 & 5 & 7 \\ \infty & \infty & \infty & 0 & 1 \\ \infty & \infty & \infty & \infty & 0 \end{bmatrix} = D$$

Sendo que os elementos p_{ij} são os pesos para ir do estado i ao estado j. Note que no exemplo 5, a matriz P é a transposta deste caso. Para o 2º passo, fazemos as operações necessárias à obtenção de P_2, ou seja, $P_2 = P \otimes P$.

$$P_2 = \begin{bmatrix} 0 & 1 & 3 & 10 & \infty \\ \infty & 0 & \infty & 9 & \infty \\ \infty & \infty & 0 & 5 & 7 \\ \infty & \infty & \infty & 0 & 1 \\ \infty & \infty & \infty & \infty & 0 \end{bmatrix} \otimes \begin{bmatrix} 0 & 1 & 3 & 10 & \infty \\ \infty & 0 & \infty & 9 & \infty \\ \infty & \infty & 0 & 5 & 7 \\ \infty & \infty & \infty & 0 & 1 \\ \infty & \infty & \infty & \infty & 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 3 & 8 & 10 \\ \infty & 0 & \infty & 9 & 10 \\ \infty & \infty & 0 & 5 & 6 \\ \infty & \infty & \infty & 0 & 1 \\ \infty & \infty & \infty & \infty & 0 \end{bmatrix}$$

Agora, precisamos verificar se as distâncias da matriz P_2 são menores que as da matriz D, ou seja, $D_2 = D \ominus P_2$

$$D_2 = \begin{bmatrix} 0 & 1 & 3 & 10 & \infty \\ \infty & 0 & \infty & 9 & \infty \\ \infty & \infty & 0 & 5 & 7 \\ \infty & \infty & \infty & 0 & 1 \\ \infty & \infty & \infty & \infty & 0 \end{bmatrix} \ominus \begin{bmatrix} 0 & 1 & 3 & 8 & 10 \\ \infty & 0 & \infty & 9 & 10 \\ \infty & \infty & 0 & 5 & 6 \\ \infty & \infty & \infty & 0 & 1 \\ \infty & \infty & \infty & \infty & 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 3 & 8 & 10 \\ \infty & 0 & \infty & 9 & 10 \\ \infty & \infty & 0 & 5 & 6 \\ \infty & \infty & \infty & 0 & 1 \\ \infty & \infty & \infty & \infty & 0 \end{bmatrix}$$

Repetindo iterativamente os procedimentos do 2º passo ao 4º passo, obtemos a matriz D_4, que contém a menor distância entre todos os estados.

$$P_4 = P \otimes P_3 = \begin{bmatrix} 0 & 1 & 3 & 10 & \infty \\ \infty & 0 & \infty & 9 & \infty \\ \infty & \infty & 0 & 5 & 7 \\ \infty & \infty & \infty & 0 & 1 \\ \infty & \infty & \infty & \infty & 0 \end{bmatrix} \otimes \begin{bmatrix} 0 & 1 & 3 & 8 & 10 \\ \infty & 0 & \infty & 9 & 10 \\ \infty & \infty & 0 & 5 & 6 \\ \infty & \infty & \infty & 0 & 1 \\ \infty & \infty & \infty & \infty & 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 3 & 8 & 9 \\ \infty & 0 & \infty & 9 & 10 \\ \infty & \infty & 0 & 5 & 6 \\ \infty & \infty & \infty & 0 & 1 \\ \infty & \infty & \infty & \infty & 0 \end{bmatrix}$$

$$D_4 = \begin{bmatrix} 0 & 1 & 3 & 8 & 9 \\ \infty & 0 & \infty & 9 & 10 \\ \infty & \infty & 0 & 5 & 6 \\ \infty & \infty & \infty & 0 & 1 \\ \infty & \infty & \infty & \infty & 0 \end{bmatrix} \ominus \begin{bmatrix} 0 & 1 & 3 & 8 & 9 \\ \infty & 0 & \infty & 9 & 10 \\ \infty & \infty & 0 & 5 & 6 \\ \infty & \infty & \infty & 0 & 1 \\ \infty & \infty & \infty & \infty & 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 3 & 8 & 9 \\ \infty & 0 & \infty & 9 & 10 \\ \infty & \infty & 0 & 5 & 6 \\ \infty & \infty & \infty & 0 & 1 \\ \infty & \infty & \infty & \infty & 0 \end{bmatrix}$$

*Agora, precisamo aplicar o algoritmo de roteamento à matriz D_4 dada pelo algoritmo de Floyd-Warshall min-plus. Como desejamos sair do vértice **a** para ir ao vértice **e**, analisaremos a 5ª linha da matriz $D_r = D_4^T$.*

$$\begin{bmatrix} 9 & 10 & 6 & 1 & 0 \end{bmatrix}$$

*Como partimos do vértice **a** e sua vizinhança consiste nos vértices **b**,**c** e **d**, precisamos verificar qual destes possui o menor valor quando somamos a distância até **e** com o peso para ir do vértice **a** aos seus vizinhos. Para o nosso caso, teremos:*

$$\boldsymbol{b} : 10 + 1 = 11$$
$$\boldsymbol{c} : 6 + 3 = 9$$
$$\boldsymbol{d} : 1 + 10 = 11$$

Como o vértice **c** *é aquele que possui o menor peso total, é a partir dele que seguiremos. Os vizinhos do vértice* **c** *são* **d** *e* **e**, *para os quais:*

$$\boldsymbol{d}: 5 + 1 = 6$$
$$\boldsymbol{e}: 0 + 7 = 7$$

O vértice **d** é aquele que possui o menor peso total, assim, será o próximo sobre o qual iteraremos. E como o vértice **d** possui apenas um vizinho, o vértice **e**, para ele migramos e obtemos a menor rota, que passa pelos vértices **acde**. Pode-se verificar que os algoritmos de Dijkstra e de Floyd-Warshall obteriam o mesmo caminho.

4.4 Implementação

A parte final deste projeto de graduação consistiu na implementação do modelo e do algoritmo desenvolvido em um robô *Pioneer P3DX*, numa configuração idêntica à da figura 4.5.

Figura 4.5: Robô *Pioneer P3DX* controlado por um computador portátil.

O computador portátil no qual o controle e o planejamento estão implementados utiliza o sistema operacional Xubuntu 12.04, que é uma distribuição Linux. Dentro do sistema operacional foi instalado o *Robot Operating System*, ROS, uma coleção de *frameworks* que possui diversas ferramentas para facilitar o desenvolvimento de *software* para robôs. O ROS possui uma estruturá em nós similar a de uma rede, permitindo fácil comunicação em tempo real de diversos processos. Também foi instalada a biblioteca ROSAria que provê os protocolos de comunicação e as diversas funções necessárias para o controle de robôs do fabricante *Pioneer*.

Com o computador devidamente configurado, foi possível passar à etapa de implementação do controlador, da descrição do mapa por meio do autômato M e do planejamento de trajetória utilizando o algoritmo de Floyd-Warshall $min - plus$.

A linguagem de programação escolhida foi o *Python 2.7*, por ser suportada pelo ROS, pelo seu amplo uso e pela facilidade de codificação; que torna o desenvolvimento mais ágil. Partindo dessa escolha e desses procedimentos, foi possível começar

o desenvolvimento dos programas necessários à implementação do projeto, considerando primeiro um ambiente sem obstáculos e, em seguida, um ambiente com obstáculos.

4.4.1 Ambiente sem obstáculos

O programa desenvolvido foi estruturado em classes, de forma a modularizar o sistema, permitindo realizar alterações pontuais mais facilmente. As classes desenvolvidas foram: **PID**, **PioneerP3DX**, **Matriz** e **Autômato**.

A classe **PID** cria uma malha de controle PID com ganhos proporcional, integral e derivativo ajustáveis e é responsável por receber o sinal de referência, coletar os dados da odometria do robô, calcular o erro e determinar o sinal de controle. Como ilustrado na figura 4.6, foi necessário implementar dois controladores PID, um para deslocamento linear e outro para o deslocamento angular, projetados conforme apresentado na seção 3.3.

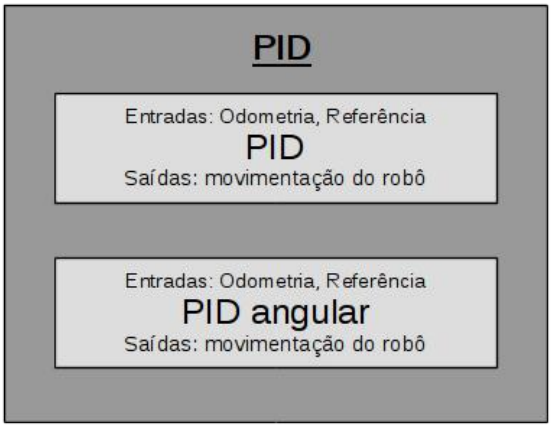


Figura 4.6: Conteúdo da classe PID.

A classe **PioneerP3DX** é responsável por representar o robô homônimo no mapa. Conforme ilustrado na figura 4.7, essa classe possui as funções necessárias à comunicação e ao controle do robô, dentre as quais, vale ressaltar:

- Função de comunicação: responsável por transmitir para o computador todos os dados odométricos de velocidade, orientação e posição;
- Função comando de velocidade: envia ao robô um comando de velocidade linear e um comando de velocidade angular.
- Função de deslocamento linear (resp. angular): utiliza a função de comando de velocidade, um objeto da classe **PID** e a posição do robô para deslocá-lo por uma distância igual a um valor de referência (resp. girá-lo por um ângulo igual ao valor de referência).

Figura 4.7: Conteúdo da classe *Pioneer P3DX*.

A classe **Matriz**, como ilustrado na figura 4.8 representa matrizes de quaisquer dimensões e possui funções que representam toda a álgebra necessária a este projeto, dentre as quais cabe ressaltar:

- Função de montagem de matrizes: função que forma a matriz de acordo com os elementos passados pelo usuário;
- Funções da álgebra matricial: coleção de funções que permite operações com matrizes, dentre elas a soma, subtração, multiplicação e transposição;
- Funções da álgebra min-plus matricial: coleção de funções que permite realizar operações da álgebra *min-plus* com matrizes, dentre elas *min*, *plus* e potenciação;
- Algoritmo de Floyd-Warshall *min-plus*: implementação do algoritmo de Floyd-Warshall *min-plus* , utilizando as funções da álgebra *min-plus* matricial.

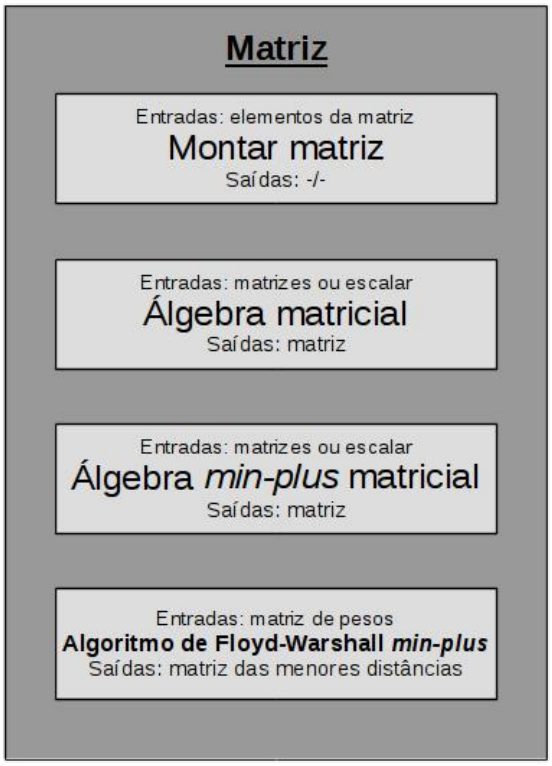


Figura 4.8: Conteúdo da classe **Matriz**

A classe **Autômato**, como mostrada na figura 4.9, é responsável por representar autômatos determinísticos, ou seja, por modelar o mapa do ambiente estruturado deste projeto. A classe possui diversas funções, dentre as quais cabe destacar:

- Função para criar o autômato: função que cria o autômato a partir da sêxtupla que define os autômatos, conforme apresentado na seção 2.1.2;
- Função de transição de estados: função que faz o autômato transitar de seu estado atual para o próximo, de acordo com o evento que ocorre;
- Função da matriz de pesos: função que gera a matriz de pesos a partir da função f, atribuindo ao elemento a_{ji} da matriz de pesos o valor igual ao peso para transitar do estado i ao estado j.

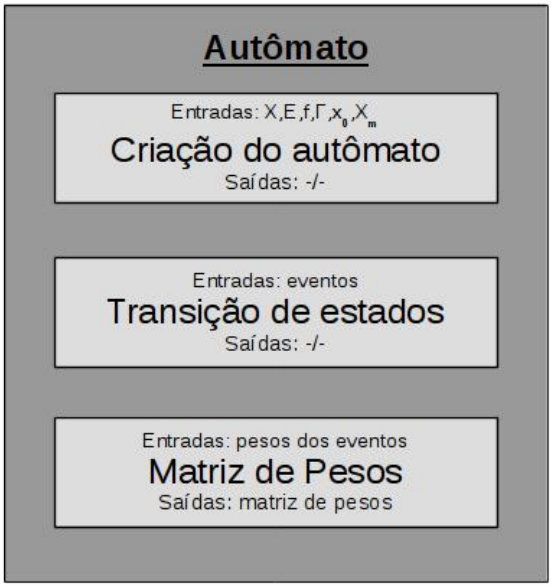


Figura 4.9: Estrutura da classe Autômato

Com todas as classes necessárias à implementação do projeto programadas, foi necessário criar uma rotina que utilizasse todas as classes e funções. O *script* **Rota min-plus**, conforme representado pela figura 4.10, possui as seguintes funções:

- Criação do robô e do autômato: rotina do *script* que cria um objeto da classe **PioneerP3DX** para estabelecer a comunicação e o controle do robô e, a partir da classe **Autômato**, cria o autômato M que representa o mapa;
- Função de alocação dos estados 39 e 40: função que aloca os estados 39 e 40, bem como os eventos necessários, gerando o autômato M';
- Algoritmo de roteamento: algoritmo que, a partir da matriz D, que armazena as menores distâncias entre os estados, determina qual sequência de estados corresponde ao menor caminho e quais eventos causam tais transições; 1
- Função de execução de tarefas: função que executa uma dada sequência de eventos correspondente ao menor caminho.

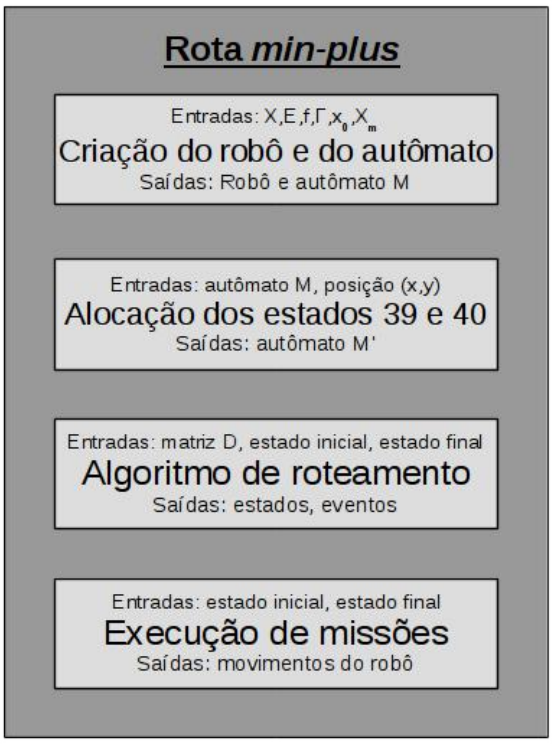


Figura 4.10: Estrutura do Programa de execução

Resultados obtidos em experimentos sem obstáculos

A seguir, apresentaremos os resultados obtidos em um experimento no qual não se considera a existência de obstáculos. Para o ambiente estruturado da figura 4.11, consideramos que o robô está em seu ponto de recarga no estado 1 e o objetivo encontra-se nas coordenadas $(5m; 5m)$ (estados 39 ou 40, destacados em vermelho na figura 4.11). Essa tarefa foi realizada em um modelo em escala 1:3,33 do ambiente estruturado da figura 4.11.

Inicialmente, o robô encontrava-se no ponto de recarga conforme ilustrado na figura 4.12a. Na primeira etapa da execução da tarefa, o algoritmo de Floyd-Warshall *min-plus* e o algoritmo de roteamento calcularam que a melhor rota para o robô passava pelos estados 1, 12, 11, 23 e 40, por meio da palavra $ida = a6.0g90a3.0a2.0$. Após o planejamento da trajetória, o robô executou cada um dos eventos da palavra ida, resultando na trajetória destacada em vermelho na figura 4.12b.

Depois de chegar ao estado 40, o robô planejou o caminho de volta por meio dos algoritmos de Floyd-Warshall *min-plus* e de roteamento. Desta forma, seguiu o caminho que passa pelos estados 40, 39, 26, 14, 13, 2 e 1; por meio da palavra $g180a2.0a3.0g - 90a6.0g180$, conforme ilustrado na figura 4.12c, concluindo assim sua tarefa.

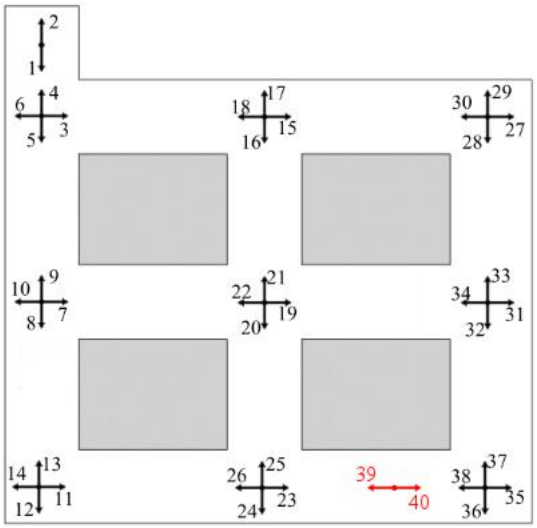
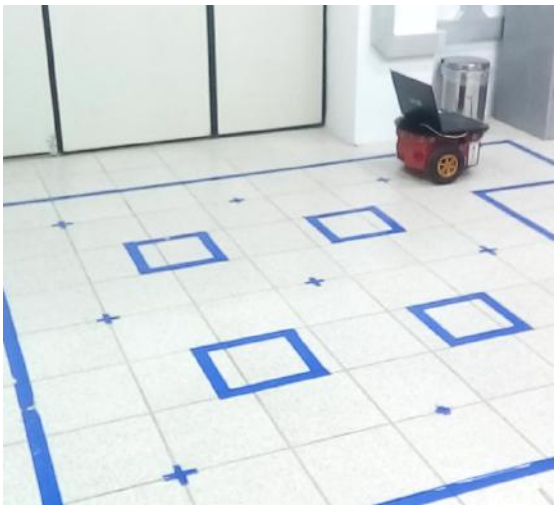
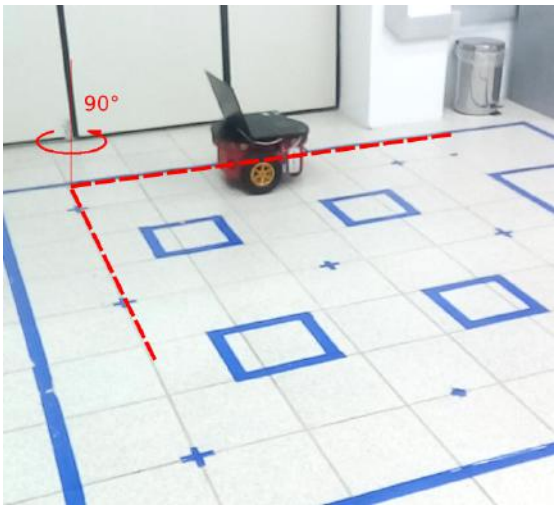


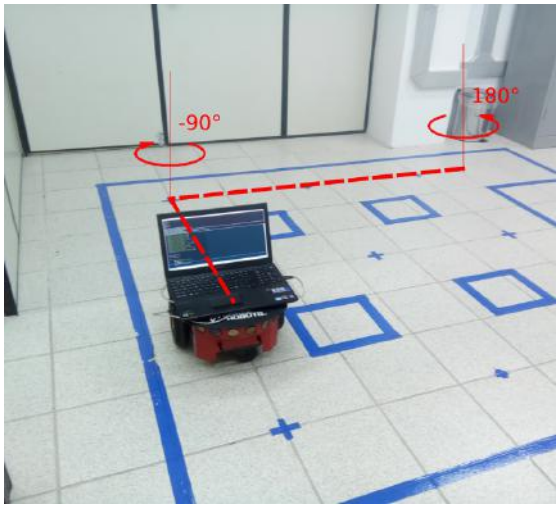
Figura 4.11: Ambiente estruturado com obstáculo.



(a) O robô na posição inicial.



(b) Trajetória de ida.



(c) Trajetória de volta.

Figura 4.12: Experimentos sem obstáculos.

O experimento, apesar de bem-sucedido, apresentou pequenos desvios na trajetória do robô, devido aos erros cumulativos apresentados pela odometria. Esse erro pode ser reduzido por meio do uso dos sonares laterais do robô, medindo a distância dele das paredes, por meio de marcos geográficos, com uma bússola ou com outros sensores.

Nesse experimento, não consideramos a existência de obstáculos. Contudo, caso uma caixa, uma prateleira, ou mesmo outro robô vier a bloquear o caminho, o robô enviado para cumprir a tarefa ficará preso, tentando transpor um obstáculo que não pode detectar. Assim sendo, na seção a seguir, faremos algumas modificações no programa original de forma a permitir o replanejamento da trajetória original quando for necessário desviar de obstáculos encontrados pelo robô durante a execução do movimento.

4.4.2 Ambiente com obstáculos

A possibilidade de objetos desconhecidos aparecerem em um ambiente estruturado, é real, seja devido a acidentes ou mesmo a sabotagem. Assim, para evitar colisões do robô com estes objetos estranhos, é preciso reconsiderar a estratégia adotada para se deslocar pelo mapa. Como o robô de modelo *Pioneer P3DX* possui sonares, decidimos utilizar 1este recurso para detectar obstáculos, evitando colisões e possibilitando um replanejamento da trajetória; o que permite cumprir o objetivo. Foram consideradas, para este projeto de graduação, somente colisões durante deslocamentos lineares, de forma que apenas os sonares frontais do robô foram utilizados.

Para utilizar os sonares, a classe **PioneerP3DX** precisou sofrer algumas mudanças. Uma função para a leitura dos sonares foi adicionada às funções já existentes. Como são consideradas colisões apenas durante deslocamentos lineares, a função guarda somente os dados obtidos pelos sonares 4 e 5, que estão localizados na parte dianteira do robô. Uma condição de parada foi adicionada à função de deslocamento linear. Caso os sonares 4 e 5 detectem um obstáculo a menos $0,5m$, o robô para imediatamente e sinaliza à rotina de execução que encontrou um obstáculo. Caso contrário, ou seja, consiga executar o evento de deslocamento, o robô sinaliza que não encontrou obstáculos, o que permite à rotina de execução passar ao próximo evento. Na figura 4.13, a estrutura da nova classe **PioneerP3DX** é ilustrada.

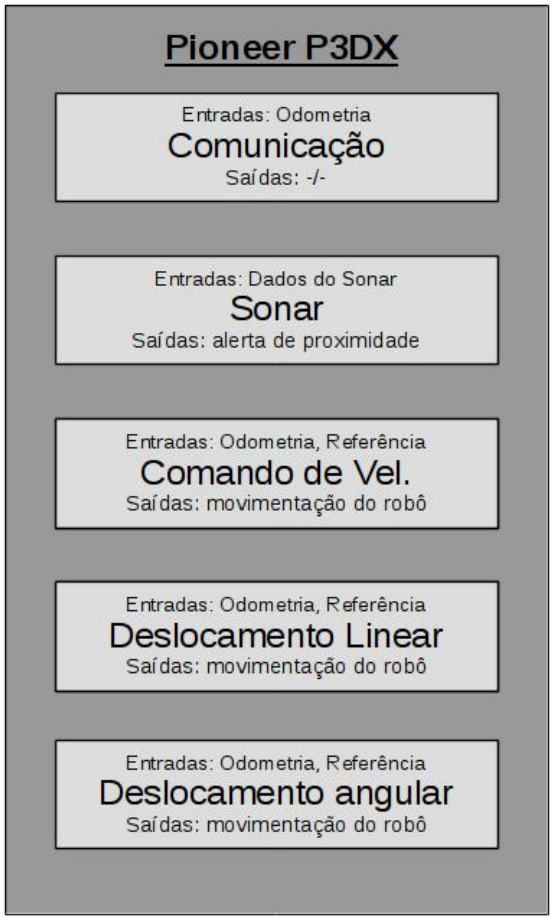


Figura 4.13: Conteúdo da classe *Pioneer P3DX* modificada.

Vale ressaltar que as classes **PID** e **Matriz** não sofreram modificações por proverem funcionalidades básicas, que não lidam com o modelo do mapa nem com o robô em si. Por sua vez, a classe **Autômato** foi modificada para contemplar a possibilidade de bloquear eventos que se tornaram impossíveis devido à presença de obstáculo no ambiente estruturado. Dados dois estados, um inicial e outro de destino, como entrada da nova função, o evento que provoca tal transição será bloqueado. Desta forma, a nova estrutura da classe **Autômato** é ilustrada pela figura 4.14.

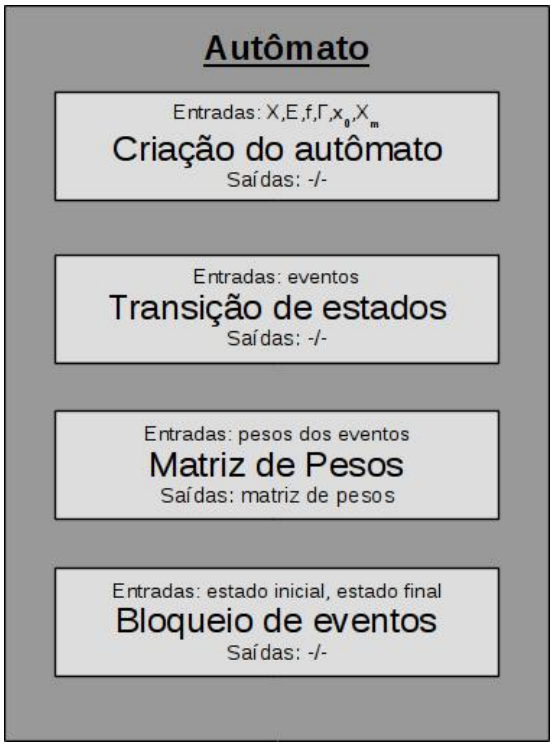


Figura 4.14: Estrutura da nova classe Autômato

Considerando as mudanças nas classes **PioneerP3DX** e **Autômato**, foi preciso modificar a rotina de execução da tarefa para considerar o replanejamento da trajetória. Durante a execução, a cada comando de deslocamento linear enviado ao robô, o programa aguarda o retorno da função de deslocamento do robô, que informa se foi encontrado um obstáculo. Caso informe o encontro de um obstáculo, o seguinte algoritmo é executado:

Algoritmo 5: algoritmo de Replanejamento de Trajetória *min-plus*

início

1. `Parar e girar o robô(180°).`
2. `Deslocar robô(distância do estado mais próximo).`
3. `Bloquear todos os eventos que levam o robô a passar pelo obstáculo.`
4. `Calcular a nova matriz de pesos` P'' `do autômato modificado.`
5. `Executar os algoritmos de Floyd-Warshall min-plus e de roteamento e reiniciar a rotina de execução.`

fim

As etapas 1 e 2 do algoritmo são comandos implementados pela própria classe **PioneerP3DX**. Quando o robô detecta um obstáculo, ele deve parar, girar 180° e

percorrer a distância até o estado mais próximo.

Para ilustrar como identificamos qual é o estado mais próximo, considere a situação apresentada na figura 4.15, na qual o robô saiu do estado 7 e deveria ir até o estado 31. Após encontrar o obstáculo, o robô para e gira 180º. A rotina de replanejamento deve identificar qual o estado mais próximo, para o qual o robô deve ir antes de replanejar. A rotina de replanejamento verifica se a distância percorrida até detectar o obstáculo é superior entre a distância do estado 7 ao 22. Nesse caso, o estado mais próximo é o estado 22. A seguir, o algoritmo precisa determinar quanto o robô precisa se deslocar até chegar ao estado 22, que é dada pela diferença entre a distância percorrida e a distância entre os estados 7 e 22.

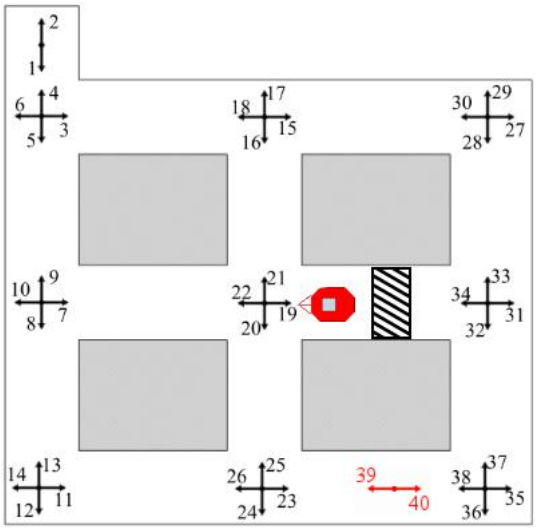


Figura 4.15: Rotina executada após a detecção de um obstáculo.

A etapa 3 do algoritmo depende da função de bloqueio de estados codificada na classe **Autômato**. A rotina analisa quais transições estão impossibilitadas pela presença do obstáculo, pois conhece a distância que o robô percorreu até encontrar o obstáculo e as distâncias entre os estados. Através da função de bloqueio, o autômato M' é transformado em M'', ao bloquearmos tais transições.

Para a etapa 4, é apenas questão de obter a matriz de pesos de M'', chamada P'' por meio da própria função da classe **Autômato**. A etapa 5, finalmente, consiste na execução dos algoritmos de Floyd-Warshall *min-plus* e de roteamento e no reinício da rotina de execução de tarefas, seguindo a melhor rota encontrada.

Resultados obtidos em experimentos com obstáculos

A seguir, iremos apresentar os resultados obtidos em um experimento no qual se considera a existência de obstáculos. Para o ambiente estruturado da figura 4.16, o ponto de partida do robô é seu ponto de recarga, no estado 1. O objetivo encontra-se nas coordenadas $(5; 5)$, representado em vermelho pelos estados 39 e

40. Há, ainda, um obstáculo desconhecido na posição $(0; 4, 5m)$, representado pelo retângulo hachurado.

Inicialmente, o robô encontrava-se no ponto de recarga conforme ilustrado na figura 4.17a. Na primeira etapa da execução da tarefa, o algoritmo de Floyd-Warshall *min-plus* e o algoritmo de roteamento calcularam que a melhor rota para o robô passava pelos estados 1, 12, 11, 23 e 40, por meio da palavra $ida = a6.0g90a3.0a2.0$, conforme indicado em vermelho na figura 4.17a. O robô tentou deslocar-se do estado 1 até o estado 12, mas encontrou o obstáculo com seu sonar após percorrer $5, 0m$, conforme indicado pela figura 4.17b. Assim, girou 180° e percorreu $1, 5m$, chegando ao estado 9, como indicado pela figura 4.17c. A partir daí, o robô removeu os eventos que o levariam a encontrar novamente o obstáculo; ou seja, os eventos que fazem o autômato transitar do estado 1 para o 12, do 5 para o 12, do 8 para o 12, do 13 para 9, do 13 para o 4 e, finalmente, do estado 13 para o 2. Foi preciso então obter a nova matriz de pesos e recalcular a melhor rota com os algoritmos de Floyd-Warshall min-plus e de roteamento, gerando a trajetória em vermelho da figura 4.17c; passando pelos estados 9, 7, 19, 20, 24, 26, 14, 13 e 39; por meio da palavra $ida2 = g - 90a3.0g - 90a2.5g - 90a3.0g - 90a1.0$.

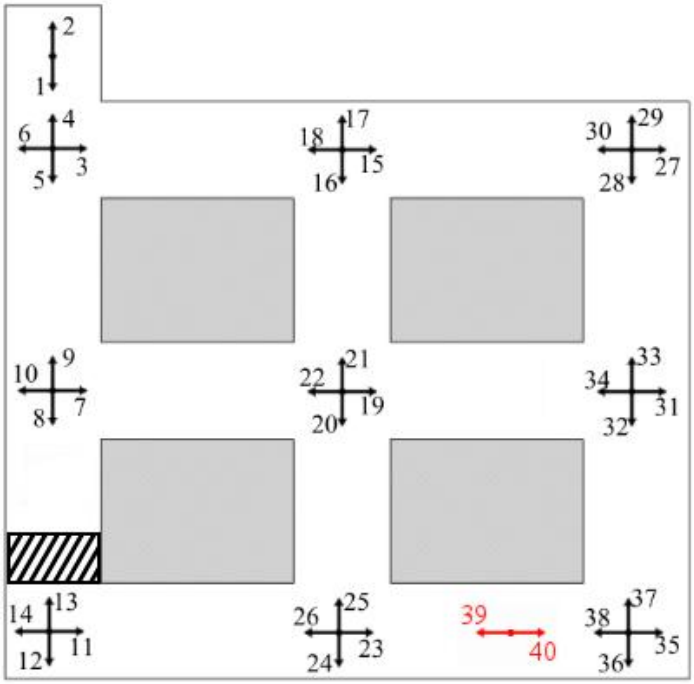
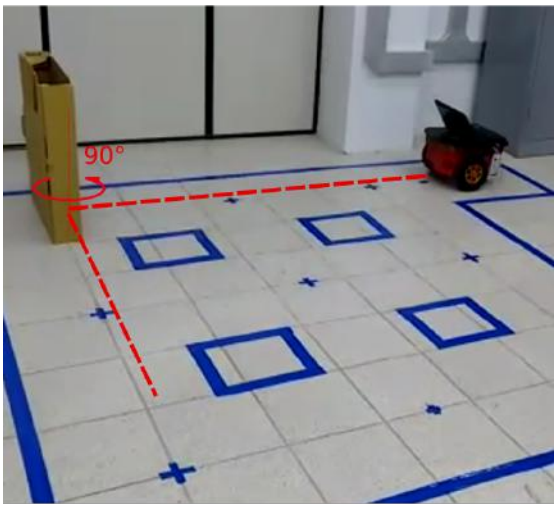
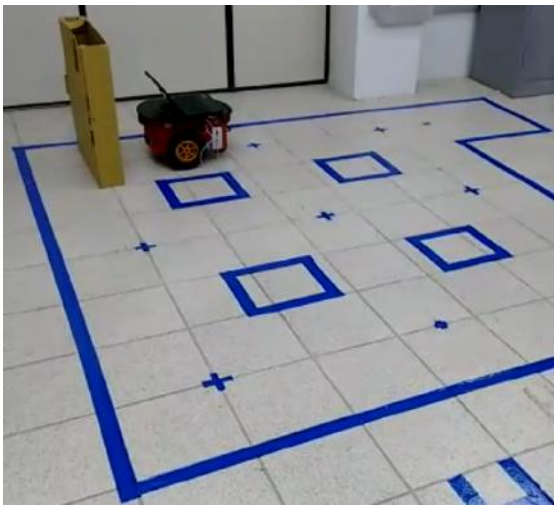


Figura 4.16: Ambiente estruturado com obstáculo.

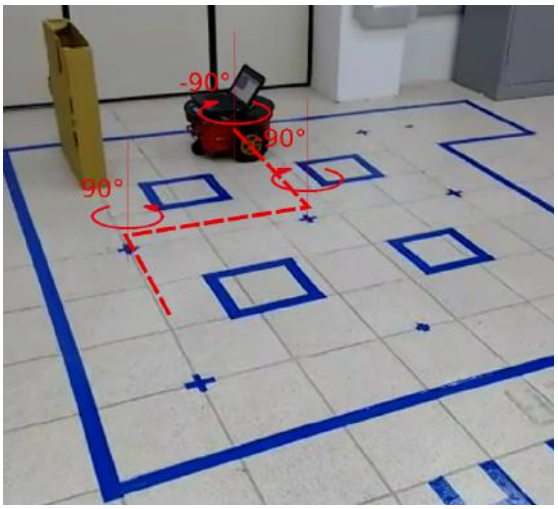
Chegando ao estado 40, o robô planejou o caminho de volta por meio dos algoritmos de Floyd-Warshall *min-plus* e de roteamento, marcado em vermelho na figura 4.17d. Seguiu, então, o caminho que passava pelos estados 39, 40, 12, 11, 23, 25, 21, 22, 10, 9, 2 e 1; concluindo sua tarefa.



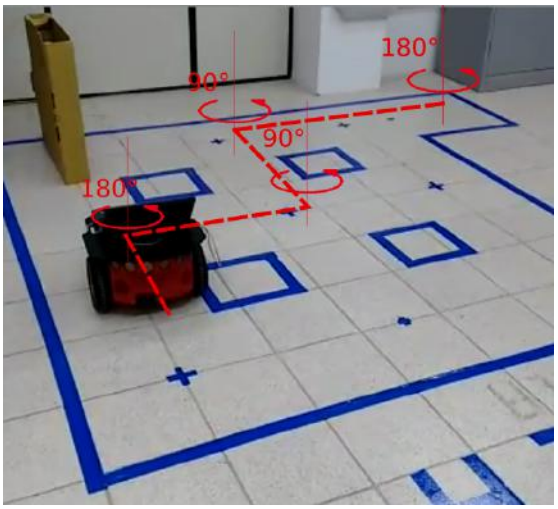
(a) Posição inicial do robô e sua trajetória.



(b) O robô encontra o obstáculo e para.



(c) O robô no estado 9 e sua trajetória replanejada (em vermelho).



(d) O robô no estado 40 e sua trajetória de retorno (em vermelho).

Por meio deste experimento, pudemos identificar certos problemas na identificação de obstáculos inerentes à natureza do sonar. Caso a superfície do obstáculo não estivesse perfeitamente alinhada à frente do robô, ou fosse plana e regular, o sonar era pouco capaz de detectar obstáculos.

Capítulo 5

Conclusões e Trabalhos Futuros

Neste trabalho foi apresentada uma abordagem para o planejamento de trajetórias de robôs móveis em ambientes estruturados, considerando a presença de obstáculos desconhecidos *a priori*. Tal abordagem consiste em modelar ambientes estruturados, por meio de um autômato determinístico, cujos estados representam determinadas posições e orientações do robô no mapa; em obter uma matriz de pesos a partir do autômato e então lançar mão do algoritmo de Floyd-Warshall *min-plus* e do algoritmo de roteamento para calcular o menor caminho entre dois estados.

Como ambientes estruturados possuem sua configuração conhecida e invariável, essa abordagem evita a grande maioria dos problemas relacionados à localização do robô no ambiente, dependendo apenas da precisão das distâncias aferidas e dos controladores para o sucesso da navegação. Como desvantagem, porém, toda vez que for necessário realizar uma mudança no ambiente será preciso criar um novo autômato para a nova configuração do ambiente.

Para que o menor caminho calculado pudesse ser percorrido pelo robô, fez-se necessário criar um sistema de controle, que consistiu em dois controladores separados. O primeiro controla o deslocamento linear do robô e o segundo controla o deslocamento angular. Antes, porém, foi preciso identificar o modelo matemático do robô. Utilizando a resposta do sistema ao degrau e o método das áreas, os modelos do sistema para o deslocamento linear e angular foram obtidos. A partir desses modelos matemáticos, lançando mão do método do lugar das raízes, foi possível sintonizar os dois controladores PD, permitindo ao robô navegar pelo ambiente estruturado.

Uma abordagem aternativa para o planejamento de trajetória seria basear a busca do menor caminho no algoritmo de Dijkstra ao invés do algoritmo de Floyd-Warshall *min-plus* . Essa nova abordagem, provavelmente resultaria em algoritmos com menor complexidade computacional. No entanto, deve-se analisar se há uma real vantagem computacional quando a tarefa é repetida várias vezes e o ambiente estruturado não é modificado.

Apesar do sistema de controle desenvolvido neste trabalho ter um bom desempe-

nho, ele apresenta erros em deslocamentos angulares. O robô *Pioneer P3DX* possui sonares ao seu redor e o ambiente estruturado possui corredores bem definidos. Assim, é possível usar os sonares para corrigir a orientação do robô, ou outros sensores, como uma bússola, um *rangefinder* laser ou uma câmera. Além disso, outro problema que pode ser estudado é o mapeamento do ambiente, para gerar automaticamente o autômato, no qual seriam utilizados os algoritmos desenvolvidos.

Referências Bibliográficas

[1] RIFKIN, J. *The end of work: the decline of the global labor force and the dawn of the post-market era*. 1st ed. 200 Madison Avenue New York, NY 10016, G.P. Putnam's Sons, 1995.

[2] WURMAN, P. R., D'ANDREA, R., MOUNTZ, M. "Coordinating Hundreds of Cooperative, Autonomous Vehicles in Warehouses", *AI Magazine*, v. 29, pp. 9–20, 2008.

[3] MOLINA, L. *Desenvolvimento de uma arquitetura de navegação deliberativa para robôs móveis utilizando a teoria de controle supervisório*. Dissertação de Mestrado, COPPE/UFRJ, Rio de Janeiro, RJ, Brasil, Março 2010.

[4] CAMPOS, D. V. "Desvio de obstáculos em trajetórias de robôs móveis com arquitetura reativa utilizando autômatos". Projeto de Graduação, UFRJ, Rio de Janeiro, RJ, Brasil, Julho 2010.

[5] DA SILVA VIANA, G. *Diagnose de falhas de sistemas a eventos discretos com transições ponderadas*. Dissertação de Mestrado, COPPE/UFRJ, Rio de Janeiro, RJ, Brasil, Março 2014.

[6] CASSANDRAS, C. G., LAFORTUNE, S. *Introduction to Discrete Event Systems*. 2nd ed. New York, Springer, 2008.

[7] EULER, L. "Solutio problematis ad geometriam situs pertinenti", *Commentarii Academiae Scientiarum Imperialis Petropolitana*, pp. 128–140, 1736.

[8] HOPKINS, B., WILSON, R. J. "The Truth about Konigsberg", *The College Mathematics Journal*, v. 35, pp. 198–207, 2004.

[9] CORMEN, T. H., LEISERSON, C. E., RIVEST, R. L., et al. *Introduction to Algorithms*. Cambridge, Massachusetts, The MIT Press, 2009.

[10] ASTRÖM, K. J., T.HAGGLÜND. *PID controllers theory, design, and tuning*. 2nd ed. New York City, United States of America, Springer, 1995.