UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE CIÊNCIA DA COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

MODELOS BASEADOS EM AUTÔMATOS CELULARES PARA O PLANEJAMENTO DE CAMINHOS EM ROBÔS AUTÔNOMOS

GIORDANO BRUNO SANTOS FERREIRA

Uberlândia - Minas Gerais
2014

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE CIÊNCIA DA COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

GIORDANO BRUNO SANTOS FERREIRA

MODELOS BASEADOS EM AUTÔMATOS CELULARES PARA O PLANEJAMENTO DE CAMINHOS EM ROBÔS AUTÔNOMOS

Dissertação de Mestrado apresentada à Faculdade de Ciência da Computação da Universidade Federal de Uberlândia, Minas Gerais, como parte dos requisitos exigidos para obtenção do título de Mestre em Ciência da Computação.

Área de concentração: Inteligência Artificial.

Orientadora:
Prof[a]. Dr[a]. Gina Maira Barbosa de Oliveira

Uberlândia, Minas Gerais
2014

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE CIÊNCIA DA COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Os abaixo assinados, por meio deste, certificam que leram e recomendam para a Faculdade de Ciência da Computação a aceitação da dissertação intitulada "**Modelos Baseados em Autômatos Celulares para o Planejamento de Caminhos em Robôs Autônomos**" por **Giordano Bruno Santos Ferreira** como parte dos requisitos exigidos para a obtenção do título de **Mestre em Ciência da Computação**.

Uberlândia, 7 de Fevereiro de 2014

Orientadora: _____

Profª. Drª. Gina Maira Barbosa de Oliveira
Universidade Federal de Uberlândia

Banca Examinadora:

_____

Prof. Dr. Carlos Roberto Lopes
Universidade Federal de Uberlândia

_____

Prof. Drª. Patrícia Amâncio Vargas
Heriot-Watt University

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE CIÊNCIA DA COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Data: Fevereiro de 2014

Autor:	**Giordano Bruno Santos Ferreira**
Título:	**Modelos Baseados em Autômatos Celulares para o Planejamento de Caminhos em Robôs Autônomos**
Faculdade:	**Faculdade de Ciência da Computação**
Grau:	**Mestrado**

_____

Autor

Dedicatória

À todas as pessoas que sonham um dia desenvolver algo relevante para a ciência.

Agradecimentos

Agradeço primeiramente aos meus pais Darci e Maria Ivanete pelo apoio em todos os momentos.

Meus irmãos Aryadna e Leonardo pelos valiosos conselhos durante toda a realização deste trabalho.

Minha namorada Lívia pela compreensão, apoio e força quando mais precisei.

À minha orientadora Dr^{a}. Gina Maira Barbosa de Oliveira pela sua ajuda, paciência, e ideias relevantes para a conclusão deste trabalho.

À CNPQ pelo apoio financeiro.

Por fim, gostaria de agradecer às pessoas que de uma forma ou de outra também foram importantes para o término desta etapa. Gabriel Santos, Matheus Aguiar, Leandro Cunha, Luís Perini, Thiago Girello, Tatiana Scramin.

A todos, meu muito obrigado!

"Viva como se fosse morrer amanhã.
Aprenda como se fosse viver para sempre."
(Mahatma Gandhi)

Resumo

No problema do planejamento de caminhos para robôs autônomos, o objetivo é encontrar uma lista de passos a serem aplicados para se obter um caminho entre o ponto inicial e a meta. Este trabalho visa a investigação e implementação de modelos baseados em autômatos celulares (ACs) para o planejamento de caminhos. Em uma fase inicial, foi realizado um estudo comparativo entre os métodos de planejamento de caminhos baseados em autômatos celulares publicados na literatura. Posteriormente, foram escolhidos dois trabalhos publicados que foram implementados em ambientes de simulação para se verificar a real aplicabilidade dos métodos propostos. O primeiro modelo parte de uma imagem capturada do ambiente de navegação e utiliza um AC para fazer o cálculo das distâncias entre as células livres e a meta. O segundo modelo utiliza os sensores do robô para identificar sua vizinhança a cada instante e utiliza regras de transição de ACs para determinar os próximos movimentos. Algumas limitações que impossibilitaram que os robôs obtivessem bons resultados em simulação foram identificadas e melhorias foram aplicadas aos modelos originais. Ao final, os dois novos modelos propostos exibiram um melhor desempenho do que seus precursores em diversos cenários. Para validar nossos resultados, dois ambientes de simulação foram empregados (V-REP e Webots), além da execução de alguns experimentos com robôs e-puck.

Palavras chave: autômatos celulares, robótica autônoma, planejamento de caminhos.

Abstract

Considering path planning problem for autonomous robots, the objective is to find a list of steps to be applied to obtain a path between the initial point and the goal. This work aims the investigation and implementation of cellular automata (CA) based models to path-planning. In an initial phase, a comparative study was conducted among the cellular automata-based methods to path-planning published in the literature. Subsequently, two published works were chosen to be implemented in simulation environments to verify the actual applicability of the proposed methods. The first model starts from an image captured from the environment and it applies a CA to perform the calculation of distances between free cells and the goal. The second model uses robot sensors to identify its neighborhood and it applies CA transition rules to determine the next movements. Some limitations which prevented the robots obtain good results in simulation were identified and improvements to the original models were applied. At the end, both new models exhibited better behaviors than their precursors in several scenarios. Aiming to validate our results, two simulation environments were employed (V-REP and Webots) and some experiments with e-puck robots were performed.

Keywords: cellular automata, autonomous robotics, path-planning.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Os robôs vem sendo mais utilizados como auxílio para a realização de tarefas monótonas ou perigosas, além de poderem realizá-las em uma qualidade ou velocidade superior àquelas conseguidas pelos humanos. Há algum tempo, esta ideia existia somente em histórias de ficção científica, contudo a cada dia se torna mais comum a existência de robôs na vida do homem. Hoje, temos robôs que desarmam bombas, limpam o chão da casa automaticamente ou mesmo exploram a superfície da lua. Contudo, na maioria dessas aplicações, os robôs necessitam que sua inteligência seja programada por seres humanos. Assim, as pesquisas na área de robótica visam dotar os robôs de comportamentos inteligentes [Perez 2009], aumentando sua autonomia e tornando-os assim cada vez mais independentes de um controlador humano.

Dentre as áreas de pesquisa que visam incrementar a autonomia de robôs, está a área de planejamento de caminhos. Este problema consiste em encontrar uma lista de movimentos discretos entre uma posição inicial e uma meta para determinada tarefa. A partir desta lista de movimentos, pode-se passar para o controle de navegação do robô que irá de fato comandar a movimentação para realizar este objetivo. Diversas técnicas têm sido investigadas nessa tarefa, destacando-se na literatura o uso de mapa de rotas [Kavraki et al. 1996], [Zhang et al. 2013], decomposição em células [Lingelbach 2004] [Ramer et al. 2013], campo potencial [Barraquand et al. 1992], [Jianjun et al. 2013], programação matemática [Schouwenaars et al. 2001]. Uma técnica recentemente investigada para o planejamento de caminhos são os autômatos celulares (ACs).

Um autômato celular é definido por seu espaço celular e sua regra de transição. O espaço celular é composto por um reticulado de células idênticas dispostas em um arranjo d-dimensional, cada uma com um padrão idêntico de conexões locais. A regra de transição fornece o estado da célula no próximo passo de tempo baseado na configuração da vizinhança atual. Todas as células do reticulado são atualizadas de acordo com esta regra a cada passo discreto de tempo.

ACs são modelos computacionais completamente discretos - no tempo, no espaço e nas variáveis. Além disso, são sistemas distribuídos espacialmente, compostos por com-

ponentes simples e idênticos, as células, que através de interações simples e locais geram um comportamento global complexo. Eles são capazes de representar fenômenos de alta complexidade ao mesmo tempo que podem ser simulados com exatidão por processadores digitais, devido à sua natureza intrinsecamente discreta. Adicionalmente, a arquitetura descentralizada dos ACs permite a elaboração de soluções altamente distribuídas para problemas normalmente abordados por algoritmos com uma rígida coordenação central. Essas características levaram os ACs a serem considerados em diferentes problemas no campo da robótica, em especial, no problema de Planejamento de Caminhos investigado nessa dissertação.

A utilização de ACs como ferramenta no planejamento de caminhos para robôs autônomos foi investigada em diferentes trabalhos na literatura. A principal meta destes trabalhos é planejar caminhos sem colisão para um ou mais agentes, dadas diferentes restrições ou premissas. Foram encontrados diferentes trabalhos sobre o tema [Shu e Buxton 1995], [Marchese 1996], [Tzionas et al. 1997], [Behring et al. 2000], [Ioannidis et al. 2008], [Rosenberg 2007] e, a partir da identificação de similaridades entre eles, fizemos o agrupamento dos mesmos em abordagens. Dentre as abordagens encontradas, selecionamos duas para realizarmos nossas investigações.

A primeira delas parte de uma imagem do ambiente onde o robô deve navegar, que é processada para identificar a posição inicial, a posição final e os obstáculos que devem ser desviados [Behring et al. 2000]. Nesta abordagem, os ACs são modelos distribuídos que efetuam duas tarefas: o alargamento dos obstáculos identificados na imagem e o cálculo de distâncias até a meta, associadas às células do ambiente. A partir das distâncias calculadas, o algoritmo busca traçar o melhor caminho entre a posição inicial do robô e a meta. Portanto, o caminho é calculado antes da navegação efetiva do ambiente e a partir de uma imagem capturada por uma câmera externa.

Na segunda abordagem, o robô não tem nenhum conhecimento prévio do ambiente e navega sabendo apenas a direção para onde deve seguir, mas deve ser capaz de se desviar de qualquer obstáculo que apareça durante sua navegação [Ioannidis et al. 2008]. Os sensores do robô são utilizados para identificar a cada passo discreto de tempo se existem obstáculos em sua vizinhança. O AC é usado como modelo para descrever regras de transição locais que representam os movimentos a serem efetuados a cada passo de tempo, de acordo com a configuração identificada da vizinhança. Adicionalmente, no trabalho que investigamos, é utilizado um time de robôs que deve buscar manter uma formação específica enquanto navega pelo ambiente, agindo de forma cooperativa. Nesse modelo, existem regras para desvio de obstáculos e regras que visam a manutenção da formação. Portanto, o caminho é planejado a medida que o time vai navegando e realizando uma leitura do ambiente através de seus sensores.

Nas duas abordagens, partimos de trabalhos e modelos propostos por outros autores. Após o estudo e análise de cada modelo, realizamos a simulação dos mesmos nos ambientes

V-Rep e Webots tendo a arquitetura de robôs e-puck como objeto da simulação. A partir das análises das simulações dos modelos originais buscamos identificar pontos passíveis de melhoria em cada modelo. Assim, nas duas abordagens, construímos novos modelos que buscam aperfeiçoar os anteriores. Na segunda abordagem, além das simulações, também foi possível realizar alguns experimentos com robôs reais do tipo e-puck.

1.1 Contribuições

Dentre os métodos de planejamento de caminhos, este trabalho visa a investigação e implementação de modelos baseados em autômatos celulares para a realização do planejamento de rotas em robôs autônomos.

A primeira contribuição desse trabalho para a área de planejamento de caminhos baseados em ACs é teórica. Foi identificada uma escassez de trabalhos comparativos sobre a área. Assim, foi feita uma revisão global sobre os modelos baseados em ACs para planejamento de caminhos previamente publicados. Depois de analisados, esses trabalhos foram agrupados em seis abordagens distintas.

A contribuição prática consistiu na seleção e investigação da real aplicabilidade de dois dos modelos já existentes através de experimentos em ambientes simulados. O primeiro deles utiliza os ACs para realizar um cálculo de rota a priori, a partir de uma imagem capturada do ambiente onde o robô deve navegar. No segundo modelo selecionado, os ACs modelam as regras de deslocamento de um time de robôs, que são aplicadas a medida que o grupo navega por um ambiente desconhecido. A partir de pontos críticos identificados nos algoritmos implementados, foram propostas melhorias significativas que geraram novos modelos de planejamento de caminhos. Para os experimentos, foram utilizados robôs móveis com duas rodas chamados e-puck. Também foram consideradas restrições cinemáticas, também chamadas restrições não-holonômicas, além de aplicação das leis da dinâmica, que influenciam na tarefa de navegação autônoma. Acreditamos que o refinamento dos modelos que identificamos como mais promissores na literatura, trouxe uma viabilidade maior para a aplicação de ACs em robótica.

1.2 Organização da Dissertação

Esta dissertação foi organizada em seis capítulos dispostos da seguinte forma. No presente capítulo apresentamos a motivação para o estudo do planejamento de caminhos para robôs móveis, especificamente o estudo de planejamento de caminhos utilizando autômatos celulares.

O segundo capítulo inicia com uma descrição dos autômatos celulares, modelo que será utilizado para o planejamento de caminhos de robôs móveis. Depois, faz-se uma revisão da área de robótica, com um enfoque na área de planejamento de caminhos.

Também falamos sobre a arquitetura robótica e-puck [École Polytechnique Fédérale de Lausanne EPFL 2013] utilizada nos experimentos. Os conceitos sobre a forma de implementação da movimentação do robô pode ser vista na Seção 2.2.3. Na Seção 2.2.4 mostramos as dois simuladores utilizados nos experimentos realizados. No Capítulo 3 é apresentada a contribuição teórica do trabalho, onde apresentamos as seis abordagens de planejamento de caminhos utilizando autômatos celulares encontradas na literatura.

O quarto capítulo mostra a investigação conduzida sobre um trabalho da abordagem que classificamos como Abordagem por Difusão da Distância à Meta, mais especificamente o trabalho de [Behring et al. 2000]. Na Seção 4.1, descrevemos detalhadamente o modelo de Behring e colaboradores. A Seção 4.2, mostra uma análise crítica do modelo quando implementado em um ambiente simulado contendo as restrições de cinemática e dinâmica, além dos pontos críticos encontrados no modelo em simulação. As modificações feitas para endereçar esses pontos geraram um novo modelo que é mostrado na Seção 4.3; os resultados em simulação do novo modelo também são mostrados. Concluímos o capítulo na Seção 4.4 com as nossas considerações sobre o algoritmo.

O quinto capítulo contém a investigação de um segundo modelo de planejamento de caminhos para robôs móveis baseado em autômatos celulares, classificado como Abordagem por Regra de Atualização Local [Ioannidis et al. 2008]. Este modelo também contém uma característica cooperativa e é descrito na Seção 5.1. Prosseguimos com a Seção 5.2 onde mostramos os resultados em simulação para o modelo com somente um robô no ambiente além dos problemas encontrados após a simulação. A Seção 5.3, contém as soluções propostas para melhorar o comportamento do robô durante os experimentos em simulação. Na Seção seguinte discutimos os experimentos realizados com o robô e-puck real para o ambiente com um único robô. Após os resultados com 1 único robô no ambiente, fizemos os experimentos com o time de robôs que pode ser visto na Seção 5.5. Na Seção 5.6 temos os resultados em simulação com o time de robôs com a nossa proposta de mudança no controle de formação.

O sexto e último capítulo discorre sobre as conclusões tiradas deste trabalho, além de propor sugestões para trabalhos futuros visando a continuidade da pesquisa de planejamento de caminhos para robôs móveis utilizando autômatos celulares.

Capítulo 2

Fundamentos Teóricos

A natureza multi-disciplinar deste trabalho demanda a definição de alguns conceitos sobre autômatos celulares e robótica para o total entendimento do trabalho desenvolvido. Estes conceitos serão apresentados inicialmente.

2.1 Autômatos Celulares

Autômatos celulares são sistemas computacionais dinâmicos, totalmente discretos (estados, espaço e tempo) e distribuídos espacialmente, contendo componentes simples e idênticos e interações locais que geram comportamentos globais. Os autômatos celulares foram propostos originalmente por von Neumann e Ulam como uma possível idealização de sistemas biológicos, com a proposta de modelar a auto-reprodução biológica [Wolfram 1983].

Um autômato celular (AC) é definido por seu espaço celular e sua regra de transição. O espaço celular é composto por um reticulado de N células idênticas dispostas em um arranjo d-dimensional, cada um com um padrão idêntico de conexões locais entre as células e com condições de contorno definidas. A regra de transição fornece o estado da célula no próximo passo de tempo baseado na configuração da vizinhança atual. Todas as células do reticulado são atualizadas de acordo com esta regra. [Oliveira 2003]

Formalmente, seja $\mathbb{Z}^d$ um reticulado de dimensão d, e Σ um conjunto finito de estados. Em um determinado tempo, a configuração do autômato é um mapeamento c: $\mathbb{Z}^d \rightarrow \Sigma$ que especifica os estados de todas as células do reticulado [Kari 2005]. O estado de cada célula i em um dado tempo t é denotado por s_i^t, onde $s_i^t \in \Sigma$ [Mitchell 1996].

O estado s_i^t da célula i, juntamente com os estados das células as quais i está conectada a uma distância r denominada raio, é chamada vizinhança η_i^t da célula i. A regra de transição do autômato celular é definida por uma função $\Phi : \Sigma^n \rightarrow \Sigma$ onde n é o tamanho da vizinhança, ou seja, $\Phi(\eta_i)$ fornece o próximo estado s_i^{t+1} para a célula i, como uma função de η_i^t [Mitchell 1996,Kari 2005]. Portanto, cada reticulado de um autômato celular contém apenas um sucessor, determinado pela aplicação da regra de transição, mas pode

possuir um número arbitrário de predecessores, conhecidos como pré-imagens. Reticulados sem nenhuma pré-imagem são conhecidos como Jardins do Éden, pois não podem ser alcançados pela evolução do autômato celular [Wuensche e Lesser 1992], somente podem ser impostos como reticulados iniciais.

2.1.1 Autômatos Celulares Elementares

Os ACs elementares são os modelos mais simples de autômatos celulares. Eles são definidos com dimensão $d = 1$, raio $r = 1$, Σ: $\{0, 1\}$. Existem $2^3 = 8$ padrões de vizinhança possíveis para uma célula, a qual cada uma pode ser mapeada em 0 ou 1. Por isso, existem $2^8 = 256$ autômatos celulares elementares diferentes [Kari 2005].

As regras dos autômatos celulares elementares foram investigadas e classificadas empiricamente em [Wolfram 1983]. Ele introduziu um esquema de nomenclatura que se tornou padrão desde então. Cada regra elementar é especificada como uma sequência de oito bits:

$$\eta(111)\ \eta(110)\ \eta(101)\ \eta(100)\ \eta(010)\ \eta(001)\ \eta(000).$$

A sequência de bits é uma expansão binária de um inteiro no intervalo 0 ... 255, e é chamado de o "Número de Wolfram" do autômato celular [Kari 2005]. A Figura 2.1 mostra a sequência de bits para a regra número 30, ou a regra 00011110 em binário. Esta numeração pode ser facilmente generalizada para uma quantidade maior de estados ou um aumento do raio.

A Figura 2.2 mostra a evolução de um autômato celular elementar, partindo de um reticulado arbitrário por um passo de tempo. A vizinhança que está sendo avaliada em cada atualização é composta pelas células marcadas em tons de cinza. Apesar de ser mostrado em forma sequencial, as atualizações de todas as células do reticulado podem ser feitas paralelamente em um único passo de tempo.

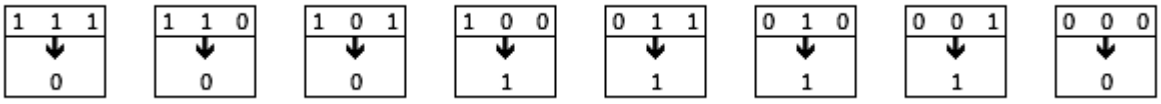


Figura 2.1: Regra de transição número 30.

2.1.2 Autômatos Celulares Bidimensionais

Quando trata-se dos autômatos celulares bidimensionais, as vizinhanças mais utilizadas são as chamadas vizinhança de Von Neumann e vizinhança de Moore [Oliveira 2003].

A vizinhança de von Neumann é definida como uma vizinhança em formato de diamante, em torno de uma dada célula (x_0, y_0), ou seja, as células que satisfazem a norma de Manhattan [Kari 2005]. Logo, a vizinhança de von Neumann contém $2r(r+1)+1$

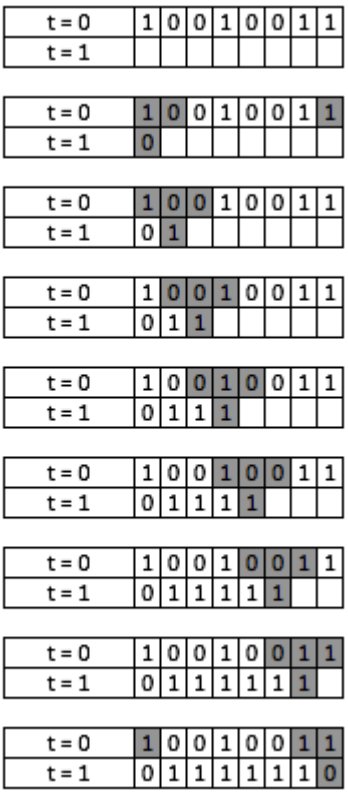
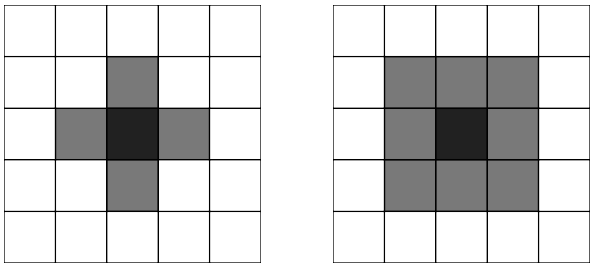


Figura 2.2: Evolução de um reticulado utilizando a regra 30, durante um passo de tempo.

células, então, para o raio 1, a vizinhança de (x_0, y_0) terá as células (x_0, y_0), (x_0-1, y_0), (x_0+1, y_0), (x_0, y_0-1) e (x_0, y_0+1).

A vizinhança de Moore é definida como uma vizinhança em formato quadrático, em torno de uma dada célula (x_0, y_0), ou seja, as células que satisfazem a norma máxima [Kari 2005]. Já a vizinhança de Moore contém $(2r + 1)^2$ células, portanto, para o raio 1, a vizinhança de (x_0, y_0) terá as células (x_0, y_0), (x_0-1, y_0-1), (x_0-1, y_0), (x_0-1, y_0+1), (x_0, y_0-1), (x_0, y_0+1), (x_0+1, y_0-1), (x_0+1, y_0) e (x_0+1, y_0+1).



(a) Vizinhança de von Neumann

(b) Vizinhança de Moore

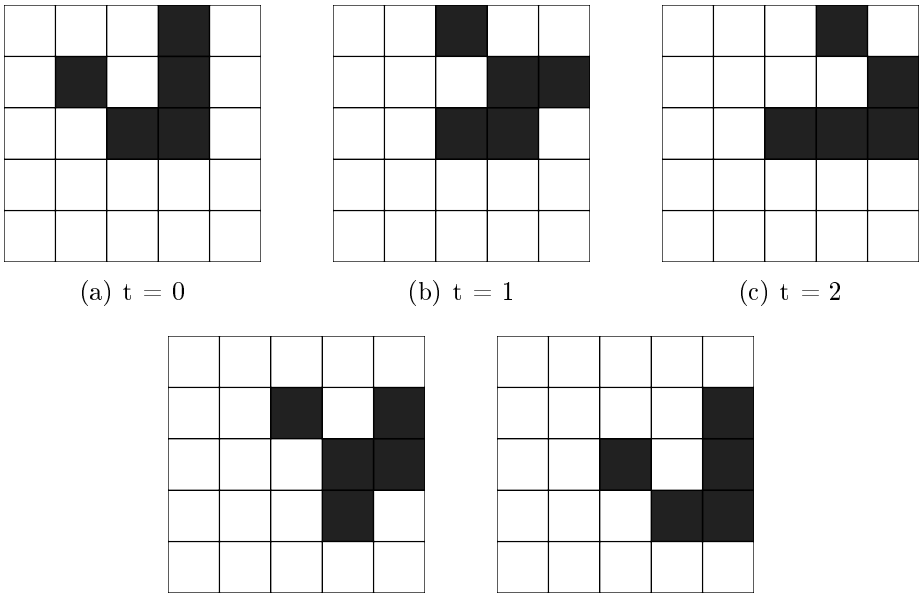
Figura 2.3: Modelos de vizinhança para autômatos celulares bidimensionais.

O exemplo mais conhecido de autômato celular bidimensional é o chamado *Game of Life*. O modelo foi proposto por John Conway em 1970 e foi primeiramente publicado em [Gardner 1970]. Consiste em um autômato celular bidimensional, de raio 1 e com

vizinhança de Moore. Esse AC possui dois estados: viva ou morta, às vezes representada por valores binários, 1 e 0. As regras de transição são as seguintes:

- Sobrevivência: cada célula viva com dois ou três vizinhos vivos, permanece viva.
- Morte: cada célula viva com quatro ou mais vizinhos vivos, morre por superpopulação; cada célula viva com um ou menos vizinhos vivos, morre por solidão.
- Nascimento: cada célula morta que tenha exatamente três vizinhos vivos, se torna uma célula viva.

Apesar das regras simples, o *Life* consegue mostrar um comportamento complexo no espaço de células. Objetos interessantes emergem com frequência após uma configuração inicial aleatória. Objetos conhecidos encontrados no *Life* são os chamados *gliders*. Eles são estruturas locais que se deslocam pelo espaço celular e são muito importantes para a construção de um autômato celular equivalente à uma máquina de Turing. A evolução temporal de um *glider* de período 4 pode ser vista na Figura 2.4, na qual as células vivas (ou estado 1) são representadas em preto, enquanto as células mortas (ou estado 0) são representadas em branco.

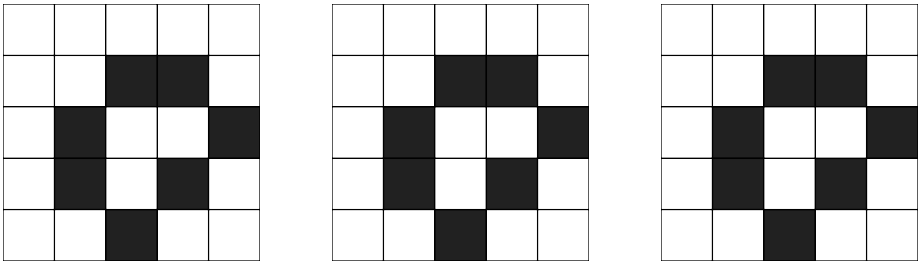


(a) t = 0 (b) t = 1 (c) t = 2

(d) t = 3 (e) t = 4

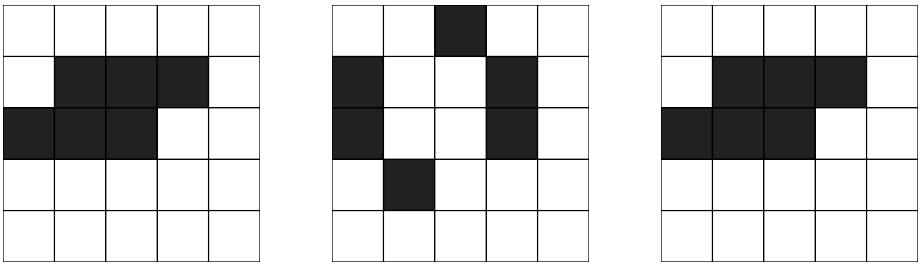
Figura 2.4: Evolução de um *glider* de período 4 no *Life*.

Outras configurações locais interessantes encontradas no *Life* são as *Still life* (Figura 2.5) que permanecem constantes nos próximos passos de tempo; os *Oscillators* (Figura 2.6) que são objetos que retornam à sua forma inicial após uma quantidade fixa de passos ≥ 2; e os *Glider guns* que são configurações que retornam à sua forma inicial após uma quantidade fixa de passos - como os *Oscillators* - além de emitir um ou mais *gliders* após cada ciclo.



(a) t = 0 (b) t = 1 (c) t = 2

Figura 2.5: Configuração *still life* no *Life*.



(a) t = 0 (b) t = 1 (c) t = 2

Figura 2.6: Evolução de um *oscillator* de período 2 no *Life*.

2.2 Robótica

A robótica refere-se ao estudo e uso de robôs. O termo foi usado primeiramente pelo autor Isaac Asimov, que publicou dezenas de livros, de variados temas, porém bastante conhecido por seus romances de ficção científica [Dowling 1995].

Um robô autônomo é definido como um robô com um sistema de controle automático, projetado para executar determinada tarefa. Um sistema é caracterizado por variáveis de entrada e saída as quais existem um relacionamento de causa e efeito entre elas. Controle é o processo de forçar uma variável de saída de um sistema para estar em conformidade com um valor desejado, denominado valor de referência. O controle pode ser realizado nas formas manual, semiautomática e automática [Unbehauen 2009].

O sistema de controle de robôs é geralmente muito complexo [Zieliński e Winiarski 2010]. Além disso, programar um sistema de controle para um robô, muitas vezes, exige conhecimentos não disponíveis durante o processo de desenvolvimento, principalmente se o ambiente o qual o robô for atuar contenha ruídos, não seja estruturado, ou seja desconhecido [Arkin 1998].

Assim, para facilitar a implementação do sistema de controle, pode-se criar um modelo de ambiente mais fácil de trabalhar, ou seja, um modelo onde o robô, o mundo e suas interações sejam feitas de forma mais simples, como por exemplo pode-se utilizar um modelo de ambiente discreto e auxiliados por algoritmos de planejamento de caminhos.

2.2.1 Planejamento de Caminhos

O planejamento de caminhos é o processo de detalhar uma tarefa em movimentos discretos. Então, podemos por exemplo planejar a movimentação entre um ponto inicial e uma meta para que não haja colisão entre o robô e obstáculos que existam pelo caminho. Este problema também é conhecido como o problema de movimentação do piano (*Piano Mover's Problem*), pois o problema é similar ao de se movimentar um piano entre duas salas sem que ele colida com as paredes e obstáculos das salas. Uma definição mais formal do problema é, dado um subconjunto U em um espaço n-dimensional e dois subconjuntos C_0 e C_1 de U, onde C_1 é derivado de C_0 por uma movimentação contínua, é possível mover C_0 até C_1 se mantendo completamente em U? [Weisstein 2013].

Este problema é bastante difícil de ser resolvido, e fortes evidências indicam que a sua resolução é de complexidade exponencial em relação aos graus de liberdade do robô [Barraquand et al. 2000]. Para um robô que pode se deslocar em um espaço bidimensional além de poder rotacionar, como o e-puck em nossos experimentos, existem 3 graus de liberdade. Assim, definido o problema, vários métodos foram propostos para a sua resolução, porém eles se encaixam em pelo menos uma das quatro abordagens genéricas principais: (i) mapa de rotas (também chamado de esqueleto); (ii) decomposição em células; (iii) campo potencial; (iv) programação matemática [Avadhanula et al.].

Abordagem por Mapa de Rotas

A abordagem por mapa de rotas consiste em capturar a conectividade do espaço livre do robô em uma rede de curvas unidimensionais chamadas "mapa de rotas" (ou *roadmap* em inglês) existentes no espaço de configuração livre. O planejamento de caminhos consiste em adicionar as posições inicial e final ao mapa de rotas e buscar um caminho entre eles [Latombe 1991]. Uma características dos métodos baseados no mapa de rotas é que geralmente eles têm a implementação simples, porém não fornecem uma boa representação do ambiente [De Souza 2008]. Como exemplos de métodos desta abordagem, temos os grafos de visibilidade, diagramas de Voronoi, silhueta e redes de sub-meta (também denominadas redes de caminho livre) [Latombe 1991].

Os grafos de visibilidade foram um dos primeiros métodos de planejamento de caminhos [Latombe 1991]. Como o nome diz, os grafos de visibilidade contém como vértices posições no ambiente e as arestas do grafo são formadas pela visibilidade entre estas posições. Estas posições relevantes para o grafo de visibilidade são definidos como as arestas de todos os obstáculos do ambiente, além da posição inicial do robô e sua meta. As arestas do grafo, são os segmentos de reta relativos a todos os pares de vértices do grafo que se encontram completamente dentro do espaço livre, ou seja, os segmentos de reta não contém uma parte dentro de um obstáculo qualquer. Assim, para realizar o planejamento, calcula-se um caminho entre os vértices contendo as posições inicial e meta

através de algum algoritmo (completo ou heurístico) de busca em grafo. Os métodos de navegação que utilizam grafos de visibilidade não possibilitam obstáculos que se movimentam no ambiente, além de ser necessário saber a localização do robô durante toda a navegação [De Souza 2008].

Dado um conjunto S de n pontos aleatórios em um plano, um diagrama de Voronoi consiste em todas as regiões de dominância para todos os pares de pontos de S. Uma região de dominância é um subconjunto do plano que esteja a uma mesma distância de dois pontos distintos pertencentes a S. Portanto, cada ponto em uma aresta é equidistante de exatamente dois pontos de S e cada vértice do diagrama é equidistante de pelo menos três pontos [Aurenhammer 1991]. A Figura 2.7 contém o resultado do cálculo de um diagrama de Voronoi para um conjunto de 8 pontos distintos. Finalmente, para realizar o planejamento de caminhos através do diagrama de Voronoi, primeiro converte-se os obstáculos em uma série de pontos que serão adicionados a S. Depois, calcula-se o diagrama de Voronoi para estes pontos e adiciona as posições inicial e meta, o caminho encontrado irá percorrer as arestas do diagrama, partindo da posição inicial até encontrar a meta. A vantagem deste método é que ele tende a maximizar a distância a qual o robô irá se movimentar em relação aos obstáculos [De Souza 2008].

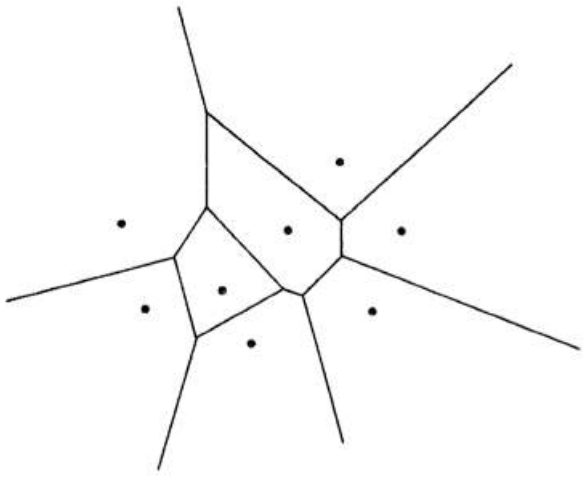


Figura 2.7: Exemplo de Diagrama de Voronoi para um conjunto S contendo 8 pontos distintos.

Embora podereosos, para dimensões maiores que 2, os métodos que utilizam grafos de visibilidade e diagrama de Voronoi contém alta complexidade [Hwang e Ahuja 1992]. Assim para ambientes diferentes do 2D, outros métodos são necessários. O método silhueta tem como objetivo reduzir a dimensão espacial do ambiente, utilizando para isso a projeção do objeto em uma dimensão superior para uma dimensão menor e então as bordas deste objeto projetado, a chamada silhueta, serão novamente projetadas até que elas se tornem linhas unidimensionais. O método da silhueta é mais utilizado como auxílio para outros algoritmos teóricos do que uma forma prática de implementação do planejamento [Hwang

e Ahuja 1992].

Diferentemente dos métodos anteriores, as redes de sub-meta não geram uma representação explícita dos obstáculos. Para o planejamento, é utilizada uma lista de configurações alcançáveis à partir da configuração inicial. Quando a configuração meta é alcançada, o método termina. As configurações alcançáveis partindo-se de outra é gerada através da aplicação de um operador, como por exemplo a movimentação do robô em linha reta. O algoritmo utiliza uma lista de configurações chamadas sub-metas e utiliza o operador para movimentar o robô através das sub-metas em uma sequência, muitas vezes fornecida por uma heurística. Caso o robô não encontre a meta, as sub-metas que foram encontradas são guardadas, pois fazem parte do caminho final, e uma nova lista de sub-metas é gerada partindo de alguma sub-meta alcançada anteriormente. A escolha de um bom operador local é determinante para definir a completude do método [Hwang e Ahuja 1992].

Abordagem por Decomposição em Células

Esta abordagem consiste em dividir o espaço livre do robô em várias regiões, idênticas ou não, chamadas células e determinar entre estas células uma relação de adjacência. Uma destas células conterá o robô e outra conterá a meta. O planejamento de caminhos irá determinar uma sequência de células adjacentes entre a célula inicial e a célula meta.

As células podem ser definidas de acordo com as bordas dos obstáculos existentes no ambiente. Este caso chama-se decomposição por dependência do objeto. Neste caso, a união das células livres define exatamente o espaço livre [Hwang e Ahuja 1992], consequentemente, caso sejam usadas técnicas de busca e de computação numérica exata, este método é completo, ou seja, sempre encontra um caminho livre, caso exista [Latombe 1991]. Um exemplo de decomposição dependente de obstáculo pode ser visto na Figura 2.8.

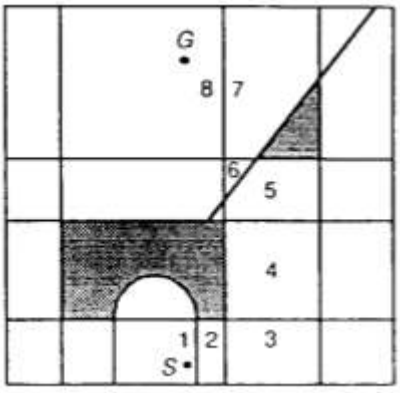


Figura 2.8: Exemplo de decomposição dependente de obstáculos. Retirado de [Hwang e Ahuja 1992].

A decomposição independente dos objetos, o ambiente é dividido em células idênti-

cas e cada célula pode conter ou não um obstáculo. Assim, as bordas dos obstáculos podem não estar perfeitamente dentro da célula. Estes pequenos erros podem ser diminuídos diminuindo o tamanho das células, tornando estes métodos "quase-completos". Esta abordagem para o planejamento de caminhos é a utilizada nos trabalhos baseados em autômatos celulares para o planejamento, como veremos posteriormente.

Dentro desta abordagem, temos um método de planejamento de caminhos conhecido como frente de onda (*wavefront*). Este método consiste em, após uma decomposição independente dos objetos, definir uma célula meta e uma célula como inicial. Depois, adiciona-se a célula meta em uma lista de células que serão visitadas. Esta meta começará com valor de distância até a meta valendo 0. O algoritmo consiste em uma busca em largura no ambiente, partindo da célula meta até encontrar a célula inicial, calculando a distância da célula até a meta. O algoritmo é assim definido: inicia-se com a célula meta na lista, retira a célula mestre e assume seu valor como 0; coloque todos os vizinhos da célula meta na lista. Agora retire uma nova célula, assuma seu valor como o de seu vizinho +1, e adicione ao final da lista os seus vizinhos que ainda não tiveram sua distância calculada. Desta forma, os vizinhos das células com valor 1 terão o valor 2, os vizinhos das células com valor 2 terão o valor 3, assim sucessivamente. O algoritmo para quando a célula com a posição inicial tem o seu valor calculado ou quando não existe mais células na lista, desta forma não existe caminho entre a posição inicial e a meta. Para encontrar o caminho, parte-se da posição inicial e escolhe as células vizinhas sempre diminuindo em 1 a sua distância, até chegar à célula com valor 0, que é a meta.

Este algoritmo, frente de onda, é bastante semelhante ao método baseado em ACs que veremos no Capítulo 4. De fato, o método utiliza um AC para o cálculo do algoritmo frente de onda de forma paralela, pois cada célula do reticulado pode ser calculada de forma independente. Porém, o trabalho nada cita sobre o algoritmo frente de onda, talvez por desconhecimento dos autores, apesar do algoritmo frente de onda ter sido proposto anos antes, como podemos ver em [Zelinsky et al. 1993] por exemplo.

Abordagem por Campo Potencial

Esta abordagem consiste em construir uma função potencial que contém seu valor mínimo na meta e um valor máximo nos obstáculos. Em todos os outros locais do espaço, a função decresce em direção à meta. Assim é possível encontrar um caminho partindo de qualquer ponto no espaço até a meta, escapando dos locais onde existem obstáculos, devido ao alto valor de sua função potencial naqueles pontos [Hwang e Ahuja 1992]. Esta denominação surgiu da metáfora que o robô é uma partícula que se movimenta sob a influência de um campo potencial produzido pela meta e os obstáculos do ambiente. Assim, a meta gera um campo potencial atrativo e os obstáculos geram um campo potencial repulsivo. A cada configuração, a direção da força é considerada a direção do movimento. A Figura 2.9 mostra um campo potencial potencial atrativo à esquerda, um campo poten-

cial repulsivo ao centro e a soma de ambos dá um campo potencial que irá ser utilizado para o planejamento de caminhos.

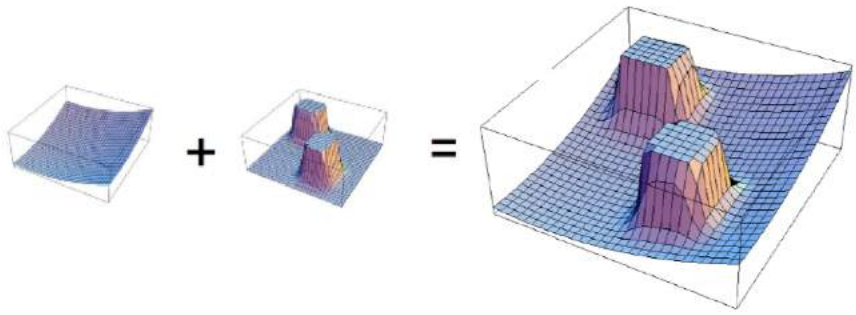


Figura 2.9: Exemplo de campo potencial gerado à partir de um ambiente contendo dois obstáculos. Retirado de [Kosecka 2013].

Um planejador que utiliza campos potenciais é simples, e rápido, contudo as funções potencial geralmente contém vários pontos de mínimos locais diferentes da meta, deixando o robô parado naquele local. Outro problema da abordagem é quando existem obstáculos convexos no ambiente, o que torna a função potencial muito pesada. Em [Hwang e Ahuja 1992], os autores afirmam que a abordagem por Campo Potencial não deve ser utilizada como um algoritmo global, mas sim como um auxiliador de outros algoritmos globais, que poderiam dividir o seu problema em pequenos problemas que poderiam ser resolvidos por campos potenciais.

Abordagem por Programação Matemática

Esta abordagem desenvolve um planejador que, através de um conjunto de inequações que representa o desvio de obstáculos, cria um modelo de otimização matemática que encontra uma curva entre as configurações inicial e final minimizando determinada quantidade. Uma vez que esta otimização é não linear e existem muitas restrições de desigualdade, um método numérico é utilizado para encontrar a solução ótima [Hwang e Ahuja 1992].

2.2.2 Arquitetura Investigada

Para os estudos na área de robótica, é necessário a existência de pelo menos um modelo de robô o qual executará os experimentos para a validação dos métodos. Assim, fez-se uma busca de arquiteturas robóticas que tivessem um embasamento na literatura, com experimentos publicados utilizando esta arquitetura, além de ser de baixo custo.

Durante as pesquisas, foram encontrados vários trabalhos na área de robótica evolutiva [Pini et al. 2007], [Capi et al. 2008], [Greeff e Nolfi 2010] que utilizavam a arquitetura e-puck [École Polytechnique Fédérale de Lausanne EPFL 2013]. Além disso, um dos trabalhos de planejamento de caminhos baseado em autômatos celulares que será detalhado

mais à frente [Ioannidis et al. 2008], também utiliza a arquitetura e-puck para os seus experimentos. Portanto, a arquitetura e-puck foi a escolhida para a validação dos experimentos que seriam realizados primeiramente em simulação.

O robô foi desenvolvido pela *École Polytechnique Fédérale de Lausanne* (EPFL) com o propósito de servir como auxiliador em cursos de engenharia. O conceito e design pode ser visto em [Mondada et al. 2009]. A Figura 2.10 mostra uma imagem do robô.

Figura 2.10: Micro robô *e-puck* utilizado nos experimentos descritos neste trabalho. [École Polytechnique Fédérale de Lausanne EPFL 2013]

Os atuadores existentes no robô são as duas rodas que contém o diâmetro próximo a 41 milímetros e com a distância entre elas em torno de 53 milímetros. A velocidade máxima de cada roda é de 1000 passos por segundo, que corresponde a 1 giro completo da roda por segundo. O robô contém também 8 sensores de infra-vermelho para verificar a proximidade com obstáculos. Estes sensores não estão dispostos de uma forma linear, existindo mais sensores na frente do robô para evitar a colisão com obstáculos à frente. A Figura 2.11 mostra a disposição espacial dos sensores em torno do robô.

Apesar de estarem em maior quantidade na parte frontal do robô, esta disposição de sensores de proximidade em todas as partes do robô, auxilia na montagem da vizinhança da célula onde o robô está localizado, conceito importante para os autômatos celulares e que foi utilizado no trabalho [Ioannidis et al. 2008] como será detalhado posteriormente.

Para o escopo deste trabalho, os únicos sensores utilizados foram os de proximidade por infra-vermelho, porém a arquitetura e-puck contém também uma câmera com resolução de 640x480 pixels, três acelerômetros colocados dentro do robô, além de microfones para a captação de sons. Como atuadores, além das duas rodas mencionadas anteriormente, o robô contém 8 leds em torno do robô e um auto-falante para geração de som. O processador é um microcontrolador da família *dsPIC*. Contém uma bateria com autonomia por entre 2 e 3 horas. A comunicação entre o robô e o computador, além da comunicação entre robôs é feita através da interface *Bluetooth*. Esta comunicação foi utilizada para os experimentos descritos no Capítulo 5.

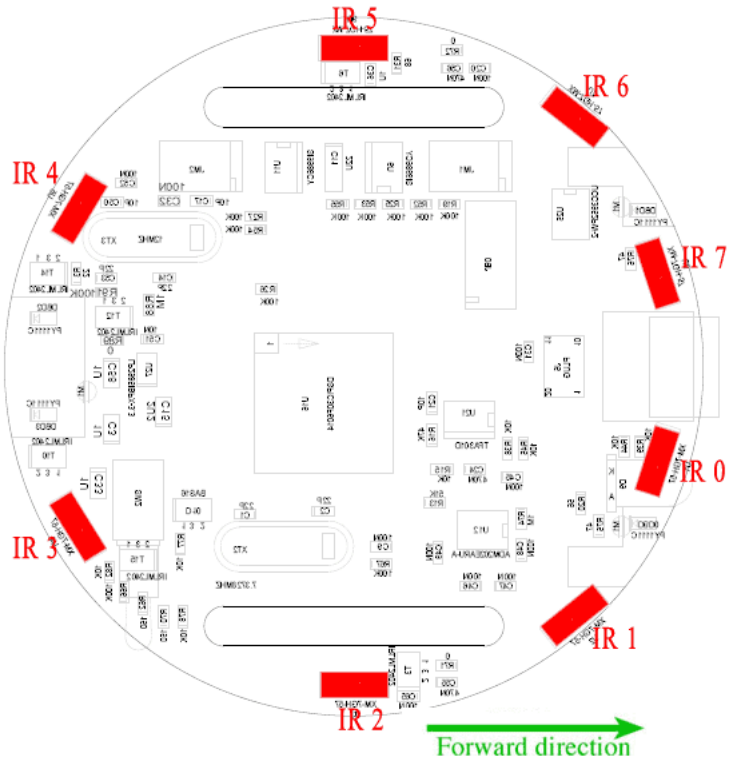


Figura 2.11: Disposição espacial dos sensores de proximidade em torno do robô. [École Polytechnique Fédérale de Lausanne EPFL 2013]

2.2.3 Implementação da Movimentação do Robô

Para que o robô se movimente (deslocamento ou rotação), é necessário implementar a forma como as rodas irão se comportar dado um determinado comando. Como o planejamento de caminhos tem como saída uma lista de movimentos discretos que serão realizados pelo robô, é necessário que o robô receba um comando de rotação para um determinada orientação, ou translação por uma distância determinada e possa realizar esta movimentação. Assim, o robô saberá em qual posição e orientação ele está em determinado momento. Duas abordagens diferentes foram implementadas em nossos experimentos. A primeira, mais simples, foi utilizada nos experimentos com o software de simulação V-Rep e é baseada na aplicação de uma velocidade constante das rodas do robô e uma contagem de tempo para definição do ângulo de rotação e da distância de deslocamento. A segunda é baseada no conceito de odometria e foi utilizada nos experimentos com o software de simulação Webots e nos experimentos com robôs reais. A seguir cada abordagem é explicada.

Movimentação por Velocidade Constante

Para a implementação do deslocamento, definimos um comportamento padrão que consiste em primeiro se fazer uma rotação visando obter a orientação desejada para em

seguida se efetuar um deslocamento até a posição desejada. Por exemplo, caso o robô esteja na direção norte, e no próximo passo de tempo deseja se fazer um movimento para o oeste, primeiro o robô faz uma rotação de 90° em direção à oeste e depois dá um passo para aquela direção.

Para se implementar de fato a rotação, determinou-se uma velocidade constante valendo $\pi/8$ rad/s em uma roda e a velocidade constante valendo $-\pi/8$ rad/s para a outra roda, desta forma, o robô consegue rotacionar em seu próprio eixo. Assim, calcula-se qual o tempo necessário para girar a quantidade necessária. Por exemplo, caso se deseje rotacionar 90°, ou $\pi/2$ rad, é necessário girar as duas rodas por 4 segundos. Com relação ao deslocamento, ou seja, a movimentação do robô para frente após a rotação para a orientação desejada, é feita se calculando a velocidade necessária para percorrer determinada distância em 1 segundo. Depois gira ambas as rodas nesta velocidade por 1 segundo e ao fim deste tempo, o robô terá percorrido determinada distância.

Como vimos, embora simples, esta forma de fazer a movimentação do robô não é a ideal. Porém os experimentos mostrados na Seção 4.2 foram realizados utilizando esta forma de movimentação, visto que o simulador V-Rep utilizado naqueles experimentos não suportava nativamente o uso de técnicas de odometria. Porém, durante os experimentos, verificou-se que o comportamento ficou próximo do que o que é encontrado de fato em experimentos com robôs reais.

Movimentação por Odometria

A odometria é um método de estimação de posição e orientação muito utilizado em robótica móvel. Apesar dos resultados serem ainda imperfeitos, a odometria retorna uma boa precisão para movimentos curtos. A ideia geral da odometria é a integração da informação do movimento incrementalmente durante o tempo em que ele se movimenta. Entretanto, a acumulação de erros de orientação causa grandes erros na estimação, erros que se tornam cada vez maiores quanto maior a distância percorrida [Junior 2008].

Para a implementação da odometria, é necessário um codificador preso nas rodas do robô que informa quantos passos de revolução foram executadas até aquele momento. Assim, podemos estimar a distância de fato percorrida pelo robô. Assim, sabendo o raio de cada roda do robô, e a quantidade de passos de revolução para uma rotação completa desta roda, podemos aplicar a fórmula: $((passos\_rev/(2 * \pi * raio\_roda)) * distancia)$ para descobrir quantos passos são necessários para percorrer determinada distância no ambiente. Para calcular a quantidade de passos para atingir determinada rotação, também é necessário a saber a distância existente entre as duas rodas do robô, sabendo-se isto, utiliza a seguinte fórmula: $((passos\_rev * base\_roda)/((360/angulo) * 2 * raio\_roda))$.

Especificamente para os robôs e-puck, é sabido que uma rotação completa da roda é feita em 1000 passos de revolução, a distância entre as rodas é de 52 milímetros e o raio das rodas é em torno de 21 milímetros.

O simulador Webots contém um codificador de odometria implementado nativamente para os robôs e-puck, assim, os experimentos mostrados no Capítulo 5 foram realizados com o auxílio da odometria para a estimação da posição e orientação dos robôs.

2.2.4 Ambientes de Simulação

Os ambientes de simulação de robôs são bastante importantes para a pesquisa na área da robótica [Tikhanoff et al. 2008], com o auxílio deles, é possível criar aplicações para teste de teorias, ideias ou eficiência de métodos, antes de embarcar a aplicação em um robô físico. Portanto, a implementação de testes em simulação antes dos testes com robôs reais é uma prática comum.

Desta forma, é possível utilizar um ambiente de simulação desenvolvido especialmente para determinado robô e determinada aplicação ou utilizar algum simulador genérico. Porém, deve-se ter em mente que um modelo de simulação não provê toda a complexidade existente em um ambiente real, além de não garantir que a aplicação seja embarcada diretamente no robô real através do ambiente de simulação [Tikhanoff et al. 2008].

Contudo, se em um segundo momento deseja-se testar a aplicação em um robô real, é recomendável a utilização de um motor de física para simular a movimentação mais real dos robôs em simulação. Alguns simuladores genéricos utilizam a ODE (*Open Dynamics Engine*) [Smith 2013], *Bullet Engine* [Coumans 2013] ou *PhysX* [NVidia 2013]. As duas primeiras são *engines* de código aberto testadas em dezenas de aplicações. A última é uma *engine* proprietária desenvolvida pela empresa *Nvidia*.

Devido ao pouco tempo de desenvolvimento deste trabalho, foi definido que seria utilizado um ambiente de simulação genérico para a execução dos testes, assim ganharia-se tempo e qualidade, visto que os simuladores genéricos são bem testados por pesquisadores ao redor do mundo e conseguem ter um resultado mais próximo de um experimento real. Entretanto, para a escolha de um ambiente de simulação para a execução dos testes deste trabalho, foi definido que seria necessário que o simulador tivesse a capacidade de simular o robô e-puck [École Polytechnique Fédérale de Lausanne EPFL 2013], que seria utilizado posteriormente nos testes com robôs reais.

O simulador utilizado nos primeiros experimentos foi o V-Rep [Robotics 2013] pois ele simulava os robôs e-puck e tinha se tornado um software de código aberto há pouco tempo. Nele é possível a implementação em 6 formas distintas: *scripts* embarcados que devem ser programados na linguagem Lua [PUC-Rio 2013]; *add-ons* também programados em Lua;*plugins* que devem ser programados em C ou C++; clientes remotos que podem ser desenvolvidos em C, C++, Python, Java, Urbi ou Matlab; ou um sistema cliente/servidor que também pode ser feito em qualquer linguagem. Para os experimentos deste trabalho foi implementado um cliente remoto por dois motivos: ser relativamente fácil de ser implementado e poder ser desenvolvido em linguagem C, podendo ser embar-

cado no robô e-puck posteriormente. O único problema desta abordagem, é o pequeno atraso que pode ocorrer entre os envios dos comandos de movimentação do robô, com isso, devido à primeira forma em que foi implementada a rotação do robô, explicada na Seção 2.2.3, - dependendo do tempo em que as rodas devem girar - este pequeno *lag* pode causar um erro que se propaga a cada nova rotação do robô. O simulador V-Rep tem a possibilidade de utilizar três motores de física para a movimentação do robô durante a simulação: ODE, *Bullet Engine*, e mais recentemente se tornou possível utilizar a *Vortex Dynamics* [CM-Labs 2013]. Foram feitos testes utilizando as duas primeiras, onde a forma de implementação da movimentação obteve melhores resultados utilizando a *engine* ODE. Cabe salientar que o simulador V-Rep possibilita que uma "imagem" do ambiente possa ser adquirida a qualquer momento, assim foi possível implementar o modelo que será descrito no Capítulo 4.

Após os testes utilizando o simulador V-Rep, o segundo modelo implementado neste trabalho e que será descrito no Capítulo 5, foi desenvolvido no simulador Webots [Cyberbotics 2013]. Isso foi possível porque após os experimentos iniciais de simulação com o V-Rep, uma licença do simulador Webots foi adquirida pelo grupo de pesquisa. A partir desse momento, decidiu-se migrar todos os experimentos para a nova plataforma. Este simulador também é capaz de simular o comportamento dos robôs e-puck, contudo com uma nova funcionalidade pois com ele é possível embarcar o código diretamente do simulador para os robôs reais, além de ser um ambiente de simulação utilizado por cerca de 1097 universidades e centros de pesquisa ao redor do mundo [Cyberbotics 2013]. Apesar de ser possível de implementação em seis linguagens (C, C++, Java, Python, Matlab e URBI), para este trabalho as implementações foram feitas em linguagem C, assim o código poderia ser embarcado nos robôs reais sem a necessidade de adaptações. O Webots utiliza o motor de dinâmica ODE que obteve resultados satisfatórios nos experimentos desenvolvidos neste trabalho. Além disso, o Webots também contém a implementação da odometria nativamente no seu modelo de e-puck. Assim os resultados da movimentação nos experimentos utilizando o Webots retornaram um erro menor do que os resultados utilizando o V-Rep.

Capítulo 3

Planejamento de Caminhos Utilizando Autômatos Celulares: Trabalhos Correlatos

A utilização de ACs como ferramenta para o planejamento de caminhos para robôs tem sido investigada recentemente. O principal objetivo é planejar caminhos livres de colisão para um ou mais agentes robóticos. Foram encontrados vários trabalhos sobre o tema [Shu e Buxton 1995], [Marchese 1996], [Marchese 2002], [Marchese 2005], [Marchese 2011], [Tzionas et al. 1997], [Behring et al. 2000], [Tavakoli et al. 2008], [Soofiyani et al. 2010] [Kostavelis et al. 2012], [Akbarimajd e Lucas 2006], [Akbarimajd e Hassanzadeh 2011], [Akbarimajd e Hassanzadeh 2012], [Ioannidis et al. 2008], [Ioannidis et al. 2011a], [Ioannidis et al. 2011b], [Rosenberg 2007], [Rosenberg 2008], [Rosenberg 2010], [Rosenberg 2012]. Após a leitura e a análise da similaridade entre esses trabalhos, agrupamos os mesmos em seis abordagens distintas: (i) Difusão de Força, (ii) Camadas e Atração para o Objetivo, (iii) Diagrama de Voronoi, (iv) Difusão da Distância à Meta, (v) Regra de Atualização Local , (vi) Envio de Mensagens. A seguir, apresentaremos cada uma das abordagens, referenciando os principais trabalhos encontrados. Ao final, apresentaremos uma analise comparativa dessas abordagens, destacando as principais características que nos levaram a selecionar duas delas para nossa investigação.

3.1 Abordagem por Difusão de Força [Shu e Buxton 1995]

A utilização de ACs em um modelo para planejamento de caminhos foi iniciada em um trabalho publicado em 1995 [Shu e Buxton 1995]. Neste trabalho, deseja-se planejar um caminho sem colisões para robôs móveis. A grande contribuição consistiu na discretização do ambiente onde o robô se moverá, em um array binário, onde 0 são as células livres e 1

as células ocupadas por um obstáculo.

No trabalho de Shu e Buxton, é apresentado como o ambiente pode ser discretizado como um array binário, bidimensional (caso o robô só seja capaz de realizar translações pelo ambiente), ou tridimensional (caso o robô também seja capaz de realizar rotações). Também é apresentado o método de busca do caminho que vai da posição inicial até a final, percorrendo os espaços livres. O método é paralelo e baseado na força de difusão dos espaços livres. Este conceito é definido como a força de ir em determinada direção da vizinhança de von Neumann, ou seja, se uma célula tem força para ir ao norte, ele pode mover um passo para o norte. Uma célula com obstáculo não adquire força para nenhuma direção.

A regra do autômato celular utilizada para o planejamento foi a seguinte: a cada passo de tempo, uma célula tem força para determinada direção se a força para aquela direção for igual a 1, ou a força para aquela direção no raio 2 for igual a 1, ou se a força para as direções não contrárias da vizinha naquela direção for igual a 1. Por exemplo: uma célula tem força para o norte se a força da célula ao norte for igual a 1, ou se a força para a célula ao norte no raio 2 for igual a 1, ou se a força das células a nordeste e noroeste tem força igual a 1.

A difusão termina em duas situações, ou se a célula inicial consegue uma força ou não tem mais espaço para difundir. No primeiro caso, o robô dá um passo em alguma direção que existe força. No segundo, conclui-se que não existe caminho até o objetivo.

No artigo, também é feita a análise do algoritmo e explicado como funcionaria caso considere também a rotação do robô, pois assim será necessário um array tridimensional.

No fim, após simulações, mostrou-se que o método é efetivo e eficiente para um ambiente estático, porém pode facilmente ser estendido para ambientes dinâmicos. Este trabalho não apresentou experimentos com robôs reais.

3.2 Abordagem por Camadas e Atração para o Objetivo [Marchese 1996, Marchese 2002, Marchese 2005, Marchese 2011]

Marchese (1996) publicou vários artigos no tema do planejamento de caminhos utilizando autômatos celulares. A característica principal da abordagem de Marchese é a utilização de um autômato celular de múltiplas camadas e robôs com restrições em sua movimentação.

O primeiro trabalho foi desenvolvido para o planejamento de caminhos para um único robô em um ambiente plano e com obstáculos conhecidos à priori. Uma informação importante para o robô é um ângulo θ que indica em qual direção está em relação à sua direção original. É proposta uma limitação de se movimentar somente para frente

e com pequenos raios de curvatura para os lados, sem a possibilidade de parar e mudar completamente sua trajetória.

A modelagem do autômato celular é baseada na arquitetura de múltiplas camadas, ou seja, um reticulado para cada variável necessária para definir a célula. O autor mostra que também é possível definir o autômato em um único reticulado, onde os estados das células são *arrays* com as variáveis necessárias. As camadas necessárias para definir cada célula são:

- Obstáculos: determina se a célula é vazia ou contém um obstáculo;
- Posição Inicial: determina se a célula é a posição inicial do robô e em qual direção o mesmo está posicionado;
- Posição Objetivo: determina se a célula é o objetivo do robô e em quais direções ele poderá chegar até esse objetivo;
- Atração para o Objetivo: mantém a distância da célula até o objetivo indo em cada uma das 8 direções da vizinhança de Moore;
- Caminho: é definido um autômato não determinístico em cada célula, pois para cada direção existe 0 ou mais direções compatíveis até o objeitvo. Esta camada contém a saída do algoritmo.

A evolução é realizada em uma camada de cada vez até encontrar um estado estacionário, obedecendo a seguinte ordem de precedência: Obstáculos, Objetivo, Inicial, Atração para o Objetivo e Extração do Caminho.

O processo de planejamento em si consiste em cinco fases. Na primeira, as células com obstáculos são aumentadas para suas vizinhas, para evitar a colisão do robô com os obstáculos, o autor elucida o fato de que é possível que espaços muito estreitos deixem de ser espaços possíveis para que o robô passe. A espessura do alargamento é definida através de um parâmetro n definido pela fórmula: $n = \text{int}(R \ / \ l) + 2$, onde R é o raio máximo do robô e l é o comprimento da célula, esta fase é mostrada na Figura 3.3. Na segunda e terceira fases, só é necessário atualizar a direção nos estados iniciais e finais.

A quarta fase é a principal e consiste em explorar o ambiente para que seja possível construir um caminho da posição inicial até o objetivo. A partir da célula objetivo, percorre-se as células vizinhas possíveis de serem alcançadas naquela rotação, incrementando a distância em 1 unidade. Esse procedimento é repetido até se chegar na célula inicial. A ideia é decidir qual a célula e a direção em que o robô precisaria estar antes de chegar à célula corrente na posição desejada, propagando até encontrar o estado inicial. Por exemplo, supondo-se que se deseja chegar ao objetivo com o robô rotacionado para o norte, com a restrição de que o robô só pode mudar de direção em 45°, as únicas células possíveis de serem predecessoras ao objetivo são: (i) a célula ao sul com o robô na direção norte, (ii) a célula ao sudeste com o robô na direção noroeste e (iii) a célula ao sudoeste

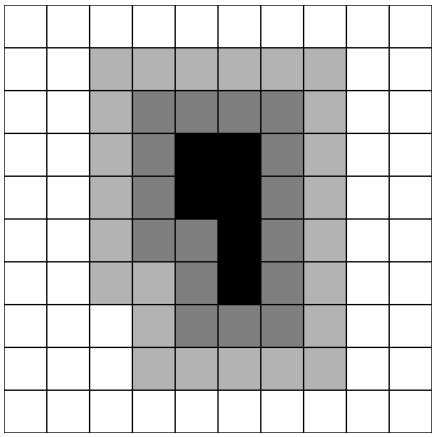
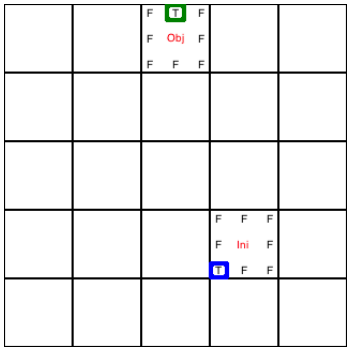


Figura 3.1: Crescimento dos obstáculos para $n = 3$. As células pretas equivalem aos obstáculos reais e as em tons de cinza equivalem ao obstáculo alargado.

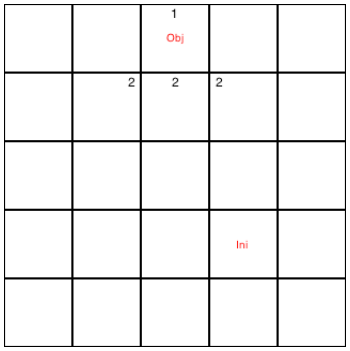
com o robô a direção nordeste. Nesse exemplo, o valor relativo a essas direções, nas respectivas células identificadas na vizinhança é incrementado em 1 unidade. Os valores relativos às demais direções nas células identificadas permanecem inalteradas.

A Figura 3.2 mostra a evolução da camada de atração para o objetivo de um cenário onde o robô deve sair da direção "sudoeste" da célula marcada por "ini" e pretende-se chegar na célula marcada por "obj" que está à três células ao norte e uma à leste (em relação à célula "ini") e em uma direção final "norte", conforme apresentado na Figura 3.2.a. O algoritmo identifica as três vizinhas possíveis de alcançar a posição desejada, considerando-se a restrição de ângulo na rotação (45°). Conforme visto na Figura 3.2.b essas vizinhas recebem o valor do objetivo (1) acrescido de 1 nas posições adequadas (ou seja, recebem 2). O algoritmo então prossegue para o próximo passo mostrado na Figura 3.2.c, onde as posições em que se é possível chegar até as posições valendo 2 recebem o valor de objetivo valendo 3. Pegando o valor 2 mais a esquerda, é possível chegar naquela célula com a orientação "nordeste", vindo da célula à esquerda na posição leste, ou da célula na diagonal inferior esquerda na posição nordeste ou na célula inferior na posição norte. Na Figura 3.2.d as posições com valor 4 são atualizadas, pois são as posições as quais se pode chegar em um objetivo 3 vindo desta. No sétimo e último passo, mostrado na Figura 3.2.e, a posição sudoeste da célula "ini" foi calculada, e tem o valor 7.

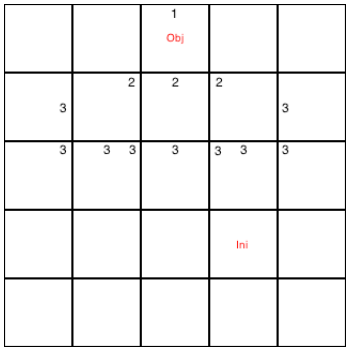
Na quinta e última fase, que utiliza os valores calculados na fase 4, parte-se da célula inicial com a orientação inicial pré-determinada, e verifica-se quais seriam as direções possíveis para o robô sair daquela célula, caso haja mais de uma possibilidade, escolhe-se a de menor atração para o objetivo, caso haja empate, faz-se uma escolha não-determinística. Esta escolha é feita até se chegar na célula objetivo com a orientação desejada. Dado o exemplo da Figura 3.2, primeiro se escolhe a célula sudoeste da célula "ini", pois é a orientação inicial do robô, nesta célula, verifica-se entre as suas posições oeste, sudoeste e sul - as posições possíveis de sair desta célula, visto que o robô está na orientação su-



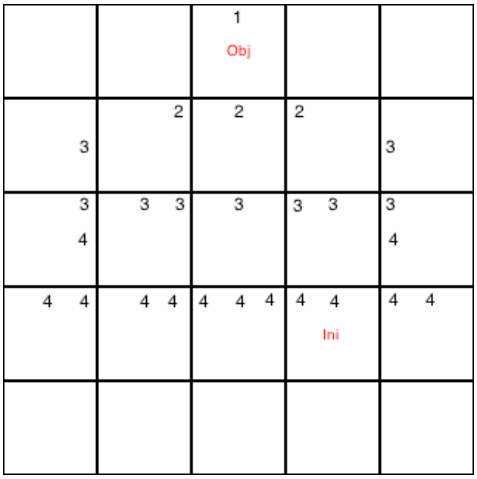
(a) Camadas de Início e Objetivo. Sendo a posição marcada em azul a orientação inicial e a posição marcada em verde a orientação final.



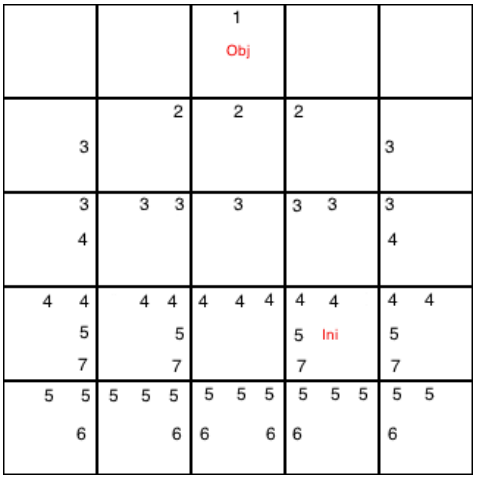
(b) Primeiro passo da Atração para o Objetivo.



(c) Segundo passo da Atração para o Objetivo.



(d) Terceiro passo da Atração para o Objetivo.



(e) Sétimo passo da Atração para o Objetivo, onde o estado inicial foi encontrado.

Figura 3.2: Evolução da camada de atração para o objetivo até encontrar o estado inicial.

doeste - aquela que tem algum valor calculado, sendo somente a célula oeste. A terceira escolha, será entre as posições noroeste, oeste e sudoeste, sendo a noroeste a única que tem o valor calculado. Ao final, o caminho encontrado é aquele apresentado na Figura 3.3. Neste exemplo específico, não existem escolhas não determinísticas, ou seja, só existe um caminho possível. Em [Marchese 1996], são apresentados os resultados de simulação do modelo, sem a utilização de robôs reais.

Em [Marchese 2002], uma extensão desse trabalho foi publicada. O robô continua a ter três graus de liberdade (x, y, θ), ou seja, o robô contém sua posição no espaço e seu ângulo de rotação. A modelagem do autômato celular baseada na arquitetura de múltiplas camadas é mantida, porém a camada de atração para o objetivo agora utiliza os pesos de cada direção, ou seja, o algoritmo foi expandido para robôs que se movimentam para a direção escolhida e não só para frente. O vetor de pesos w contém o custo de ir para: frente, diagonal para frente, mudança de direção, parada, rotação, trás, diagonal

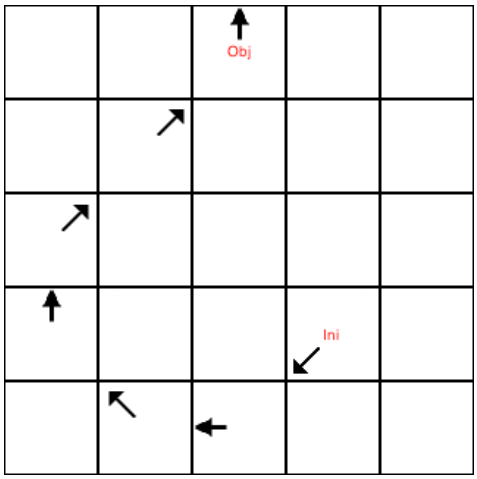


Figura 3.3: Caminhos equivalentes encontrados pelo método [Marchese 1996].

para trás. Quanto menor o peso, mais fácil para o robô aplicar determinado movimento. Logo, para um robô que só anda para frente, como os do trabalho anterior, ele pode ter um vetor w = (2, 3, 1, 0, Alto, Alto, Alto). Já um robô que contém os movimentos de um carro de passeio, pode ter os valores do vetor w = (2, 3, 1, 0, Alto, 2, 3). Assim, a regra de transição da camada de atração para o objetivo contém também o custo do movimento. Em [Marchese 2002], o autor mostra os resultados dos experimentos de simulação, também acrescentando mais de um objetivo e mais de uma posição inicial. Ao final do artigo, um estudo sobre a eficiência da atualização das camadas é feito auxiliado por uma análise simples do algoritmo. O autor demonstra que o algoritmo é de ordem quadrática.

Continuando os estudos sobre planejamento de caminhos utilizando autômatos celulares, Marchese publicou uma nova extensão do trabalho em [Marchese 2005]. Neste trabalho foi acrescentada a utilização de terrenos variáveis ao modelo, além de utilizar robôs de qualquer formato. O autômato celular multi-camadas continua sendo utilizado, porém foi acrescentada mais uma camada relativa aos terrenos. A camada de terrenos consiste em um "mapa de elevação"discretizado de forma que corresponda às células do ambiente. Na camada de obstáculos, também foi acrescentado um novo tipo de obstáculo denominado "obstáculo direcional", que é um obstáculo onde sua transposição só pode acontecer em determinada direção, e não na direção contrária. Um exemplo deste tipo de obstáculo são as portas de emergência. Para o cálculo da camada de atração para o objetivo, o custo do movimento que era baseado anteriormente no vetor w, também terá que utilizar o custo de movimento no terreno que não é mais constante como em um espaço euclidiano. O custo de movimento depende da distância 3D entre os pontos na superfície, sendo necessário acrescentar o gradiente da superfície no cálculo do custo. Após as simulações, mostrou-se que os autômatos celulares são bons formalismos para modelar o ambiente. O algoritmo é completo pois encontra todas as trajetórias livres de colisão utilizando determinada métrica, é bastante flexível pois pode ser utilizado em robôs com

diferentes tipos de movimentos e formatos e é reativo, pois se adapta às modificações no ambiente.

Em [Marchese 2008], foi publicada outra extensão do trabalho cujo o objetivo foi aplicar o método em um ambiente com vários robôs. Ele utiliza um robô coordenador que planejará os movimentos de todos os robôs para que não haja colisões entre eles e o ambiente. Os robôs podem ser de formas diferentes, porém todos devem manter a mesma arquitetura interna, que consiste em um módulo localizador, um módulo navegador, um módulo comunicador e um módulo planejador. A diferença é que um ou mais robôs serão modificados para controlar o comportamento dos outros, comunicando sempre o que foi planejado. Esta modelagem ainda encontra um grande problema que é a coordenação do movimento dos n agentes em tempo real. O modelo do autômato celular também é de múltiplas camadas, porém com diferenças quanto ao modelo em [Marchese 2002]. Nele existem duas camadas principais: a camada de obstáculos e a camada de atração. A camada de obstáculos contém 3 dimensões, x, y e tempo. A camada de atração contém 5 dimensões, espaço de configuração, (x, y, θ), robôs e o tempo. A camada de obstáculos mapeia os obstáculos do ambiente criando dinamicamente um campo de repulsão que mantém os robôs longe deles. Como o modelo é para múltiplos robôs, o planejador trata cada um deles como um obstáculo móvel para os outros robôs. Para isso, adiciona-se a dimensão do tempo. A camada de atração agora também acrescenta uma dimensão relativa ao tempo. Para o robô planejador, cada um dos outros robôs tem as suas camadas de atração que são recalculadas a cada passo de tempo. Para decidir o caminho de cada robô, escolhe-se a ordem de prioridade entre eles, e faz-se o caminho relativo àquele robô. A cada passo de tempo, a forma do robô é adicionada na camada de obstáculos, fazendo com que os próximos robôs não colidam com o primeiro. Nas simulações, foi exposto que o algoritmo não é completo. Um contraexemplo é mostrado na Figura 3.4 onde não pode ser encontrada uma solução para o problema, utilizando o robô cinza claro com prioridade sobre o robô cinza escuro. Porém tem como vantagem ser um dos poucos métodos que pode ser aplicado em robôs com diferentes formas. Também não foram realizados experimentos com robôs reais.

O último trabalho encontrado nesta abordagem é [Marchese 2011]. Aqui também é tratado o planejamento de caminhos para um conjunto heterogêneo de robôs, evitando colisões com obstáculos e com outros robôs. Os robôs também são tratados através de uma prioridade. Os de maior prioridade são planejados primeiro e os outros os veem no ambiente como obstáculos móveis. Neste trabalho, a modelagem discreta do ambiente e dos robôs é melhor detalhada. Mas a maior contribuição do trabalho é a prova da dualidade do problema de planejamento. Ou seja, o problema pode ser resolvido da célula inicial até o objetivo ou vice-versa. Com isso, pode-se aplicar uma busca bidirecional para reduzir o tempo de planejamento, encontrando a mesma solução. Também não foram realizados experimentos com robôs reais.

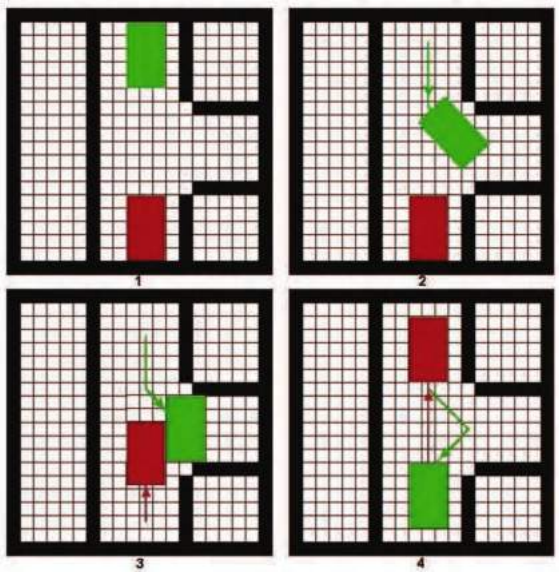


Figura 3.4: Contraexemplo onde não é encontrada solução [Marchese 2008].

3.3 Abordagem por Diagrama de Voronoi [Tzionas et al. 1997]

Em 1997 foi proposta uma outra abordagem de utilização de autômatos celulares bidimensionais para o planejamento de caminhos sem colisões para robôs [Tzionas et al. 1997]. Uma característica desta abordagem é a não necessidade de se encontrar o caminho mais curto entre uma posição inicial e uma meta. O que é desejado é um caminho o mais distante possível dos obstáculos do ambiente.

Para este trabalho, foram utilizados robôs no formato de diamante e obstáculos de qualquer formato. Estes obstáculos são representados como uma imagem discreta de todo o ambiente. O algoritmo é baseado na retração do espaço livre no diagrama de Voronoi, que é construído através da evolução temporal de um autômato celular. Essa evolução é feita após uma fase inicial onde as bordas dos obstáculos são identificadas e codificadas de acordo com sua orientação.

A arquitetura dos autômatos celulares utilizada no trabalho é um reticulado bidimensional, com vizinhança de von Neumann. Para a primeira fase, de detecção de bordas, aplica-se uma iteração do autômato celular na imagem do ambiente adquirida de forma binária. Assim, em cada célula são aplicadas doze regras distintas, chamadas *templates*. O resultado da aplicação das doze regras para cada célula, dará um reticulado com valores entre um e doze, sendo os valores 1 e 2 regiões homogêneas e 3 e 4 regiões com ruídos do tipo "sal e pimenta", estes quatro casos serão descartados na próxima fase do algoritmo.

Na segunda fase do algoritmo, é construído o diagrama de Voronoi, partindo-se do reticulado gerado na primeira fase, descartadas as regiões homogêneas e de ruído. Assim, os pontos de origem do diagrama de Voronoi, serão aqueles próximos às bordas dos

obstáculos, fazendo a expansão para os espaços livres do reticulado. O algoritmo utiliza uma *flag* para cada célula, que informa se aquela célula já teve o seu valor calculado. Essa *flag* inicia-se como verdadeira para os pontos de origem do diagrama de Voronoi, e falsa para as outras células. O algoritmo consiste em verificar para cada célula se existe algum vizinho com sua *flag* com valor verdadeiro, caso exista somente um vizinho com a *flag* verdadeira, esta célula assume o valor do seu vizinho e sua própria *flag* se torna verdadeira, neste caso, houve a expansão do espaço livre. Porém, caso mais de um vizinho tenha a *flag* com valor verdadeiro, deve-se então comparar o valor destas células, caso sejam diferentes, isto indica que a expansão veio de duas bordas distintas e se "chocaram" nesta célula, logo esta célula pertencerá ao diagrama de Voronoi. Deve-se então guardar em outra variável Vor(i,j) a iteração em que a célula foi adicionada ao diagrama, este valor será utilizado para encontrar o caminho em que o robô irá se movimentar.

Tendo as células pertencentes ao diagrama de Voronoi, o caminho que o robô percorrerá é aquele pertencentes às bordas do diagrama, ou seja, as células que contém o valor de Vor(i,j) calculado. Na Figura 3.5, esta células são aquelas numeradas. Como deseja-se que o robô percorra os caminhos mais distantes das bordas dos obstáculos, só é necessário verificar se o tamanho do robô é menor que a largura do espaço entre os obstáculos, isto é feito verificando se a diagonal do robô mais 1/2 é menor ou igual ao valor dá célula a qual o robô gostaria de se movimentar, logo, o robô percorrerá todas as células do diagrama que tenham o valor maior que sua diagonal mais 1/2. Consequentemente, o robô percorrerá o caminho mais distante de todas as bordas dos obstáculos.

Os autores implementaram o algoritmo em um chip paralelo que comprovou a eficácia do método, porém sem experimentos com robôs reais.

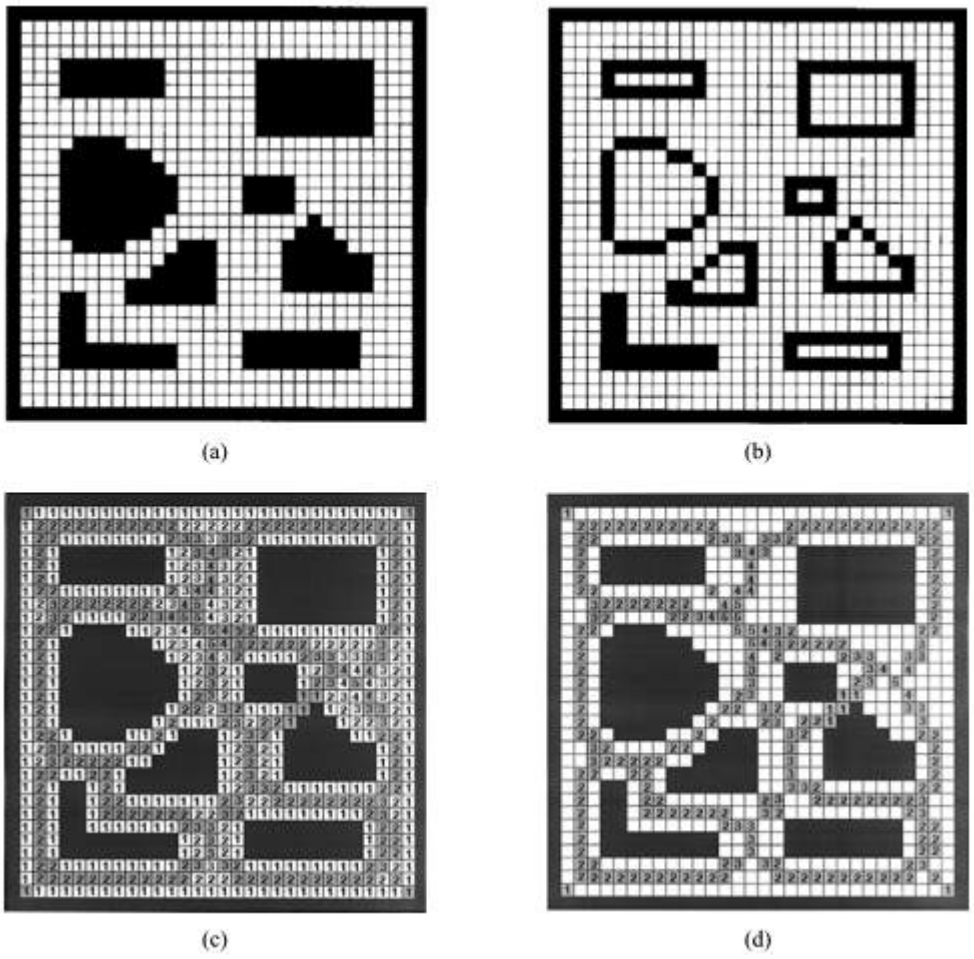


Figura 3.5: Processo de geração do caminho do modelo. (a) Imagem com a configuração dos obstáculos. (b) Fase de detecção de bordas. (c) Final da aplicação do AC, onde as células mais escuras são as pertencentes ao Diagrama de Voronoi. (d) Caminho mais distante de todos os obstáculos. [Tzionas et al. 1997].

3.4 Abordagem por Difusão da Distância à Meta [Behring et al. 2000, Tavakoli et al. 2008, Soofiyani et al. 2010, Kostavelis et al. 2012]

O primeiro artigo desta abordagem, é bastante citado como exemplo de planejamento de caminhos utilizando autômatos celulares [Behring et al. 2000]. A característica principal desta abordagem é a sua simplicidade, pois o planejamento consiste no cálculo da distância - com o auxílio de um autômato celular - entre o ponto inicial e a meta. Neste primeiro trabalho, o robô é considerado um único ponto sem orientação e sem aplicação das leis da dinâmica e da cinemática. Sendo assim, no modelo um movimento de um passo em determinada direção acontece sempre da forma perfeita. O ambiente é definido como um espaço bidimensional discretizado em células quadradas, sendo a célula definida de acordo com o tamanho do robô empregado.

O primeiro passo do algoritmo é receber como entrada uma imagem contendo a posição de cada obstáculo no ambiente e transformar em um espaço celular com quatro possíveis estados: Livre, Obstáculo, Posição Inicial e Objetivo.

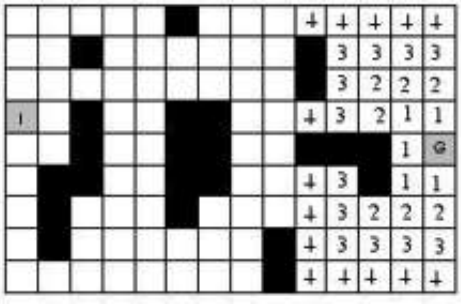
A primeira etapa onde se aplica um autômato celular tem como objetivo o crescimento das bordas dos obstáculos, para que fisicamente - com a regência das leis da dinâmica e cinemática - o robô não colida com os obstáculos. O modelo de AC emprega a vizinhança de Moore e a regra de transição consiste em: se a célula estiver no estado Livre e algum dos seus vizinhos for um obstáculo, ela se transforma em Obstáculo; caso contrário, mantém o seu estado no último passo de tempo. No trabalho [Behring et al. 2000] a regra de transição foi aplicada por quatro passos de tempo, o que resultou no alargamento dos obstáculos em uma espessura de 4 células.

A segunda etapa também aplica um AC com vizinhança de Moore e tem como objetivo associar valores de distância em relação à célula objetivo às células do reticulado. Para isso, além dos quatro estados da primeira etapa, temos como estado possível a distância entre o objetivo e a célula. Como regra de transição, caso seja uma célula livre, e exista algum vizinho que já teve sua distância calculada, o valor da célula é alterado para o valor do vizinho mais um; caso contrário, mantém o estado da célula. Este AC faz com que a distância seja calculada do objetivo até encontrar a posição inicial, ou então até não encontrar mais células livres. Este último caso ocorre apenas caso não haja caminho sem colisões entre a posição inicial e o objetivo. Dessa forma, existe uma difusão da distância desde a meta (distância igual a zero) até que a célula da posição inicial seja alcançada, como exemplifica a Figura 3.6.

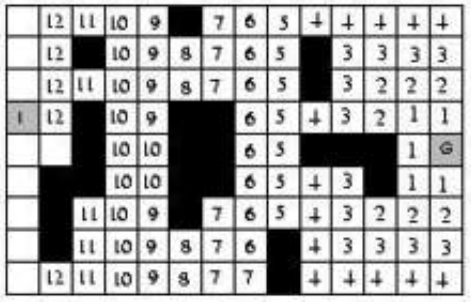
No primeiro passo do algoritmo, as vizinhas da célula contendo o "G" tem o valor igual a 1. No segundo passo, as vizinhas das células contendo o valor 1 tem o seu valor calculado igual a 2. No terceiro passo do algoritmo, as vizinhas das células contendo o valor 2 terão o valor 3 calculado. No quarto passo, temos a Figura 3.6.a, onde até as células com valor 4 foram calculadas. Com a difusão da distância prosseguindo, na décima terceira iteração, as vizinhas das células contendo o valor 12 seriam calculadas, só que no exemplo da Figura 3.6, a difusão encontra a célula inicial, que contém o valor "I", terminando o algoritmo. O final do processo pode ser visto na Figura 3.6.b.

Em uma etapa final, um caminho entre a posição inicial e o objetivo é escolhido. Qualquer caminho saindo da posição inicial até o objetivo, sempre diminuindo em um a sua distância a cada passo, é um caminho válido. Assim, o robô escolherá algum dos caminhos possíveis para o planejamento. A Figura 3.6.c apresenta um possível caminho encontrado pelo algoritmo no exemplo ilustrado na Figura 3.6.b.

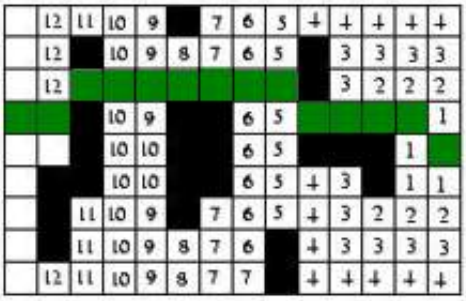
Os experimentos que comprovaram a viabilidade do método foram feitos utilizando um robô real em uma mesa de teste baseada nas especificações da RoboCup, e uma câmera fotográfica que tirava fotos do ambiente para que o caminho fosse planejado. Porém, no trabalho só foi mostrado que o robô planeja um caminho até a meta, não mostrando de



(a) Quarta iteração do algoritmo.



(b) Décima-terceira iteração.



(c) Caminho encontrado pelo algoritmo.

Figura 3.6: Distância do objetivo até cada célula e caminho encontrado pelo algoritmo.

fato a movimentação do robô seguindo as leis da dinâmica. Esse modelo será descrito mais detalhadamente no Capítulo 4.

Em 2008 um trabalho baseado em [Behring et al. 2000] foi publicado [Tavakoli et al. 2008]. Duas contribuições foram feitas, a primeira foi a utilização de vários robôs no mesmo ambiente e compartilhando um mesmo objetivo, visto que um ambiente com vários robôs tende a existir maior número de colisões entre os robôs; a segunda foi a inserção de custos na movimentação, pois uma movimentação pela diagonal é mais custosa do que uma movimentação em linha reta.

O conceito de controle multi-agente é citado neste trabalho. Este controle pode ser realizado de forma centralizada ou distribuída. Neste trabalho eles utilizaram a forma centralizada, ou seja, um único controle determina a movimentação de todos os agentes.

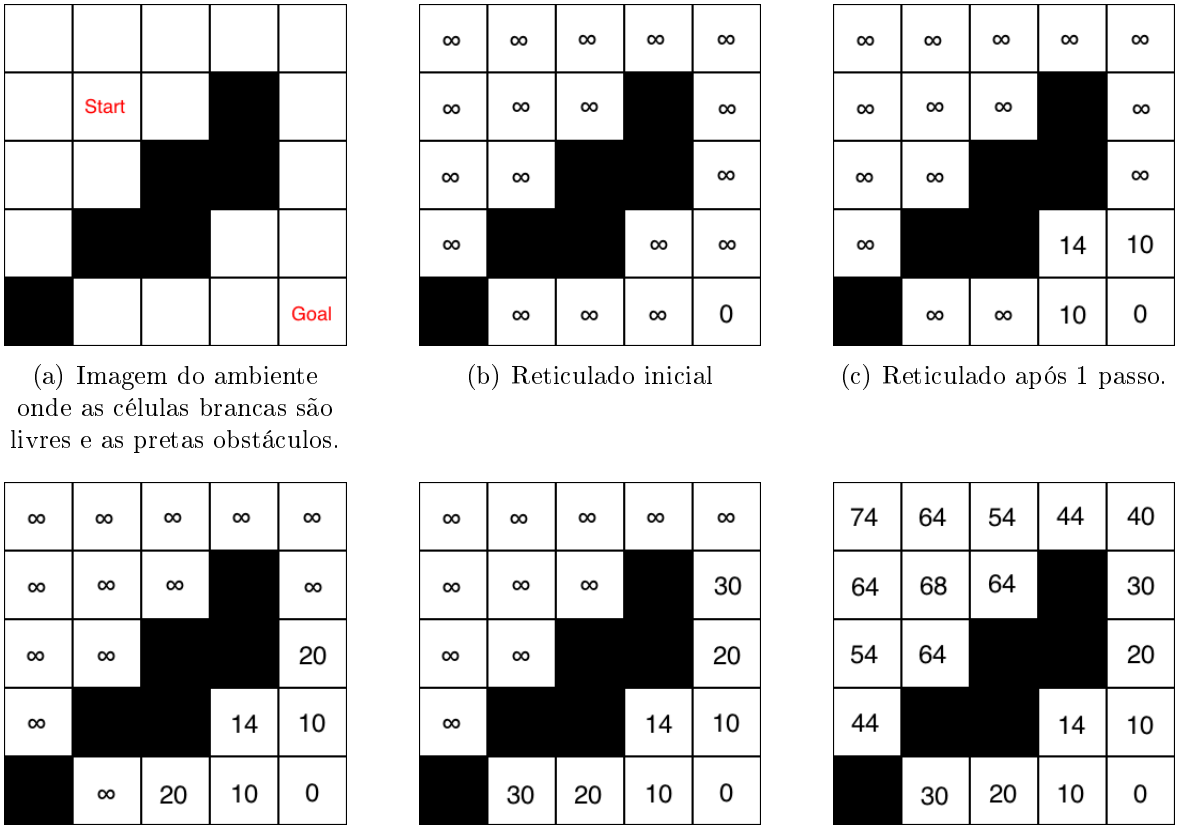
O ambiente do problema também é uma grade em formato quadrático, sendo cada célula representada nos sistemas de coordenadas cartesianas. Cada célula contém um dos três objetos: Agente, Obstáculo ou Objetivo. Se uma célula contém um agente ou obstáculo então é uma célula ocupada, caso contrário é uma célula livre.

Para calcular a distância relativa entre as células, assumiu-se que um movimento reto tem o custo de 10 e um movimento diagonal tem o custo de 14 passos de tempo, aproximando assim da norma euclidiana.

Os passos de tempo são muito importantes no algoritmo, pois eles definem em quais os momentos cada robô pode chegar a determinada célula. Exemplo: se um robô está em uma célula (x,y) no tempo t=0, ele pode chegar na célula (x+1,y) no tempo t=10, e na

célula (x+1,y+1) no tempo t=14 caso ele ande na diagonal, ou no tempo t=20 caso ele faça dois passos retos. Caso um agente ocupe a célula objetivo em um passo de tempo, ele o desocupará no próximo.

Também é utilizada a vizinhança de Moore, e todas as células livres começam com distância igual a infinito. A regra de transição é definida como: se a célula corrente for livre, a distância é definida como o menor valor entre o seu valor corrente, os valores em linha reta mais 10 e os valores na diagonal mais 14, como mostrado na Figura 3.7.



(a) Imagem do ambiente onde as células brancas são livres e as pretas obstáculos.

(b) Reticulado inicial

(c) Reticulado após 1 passo.

(d) Reticulado após 2 passos.

(e) Reticulado após 3 passos.

(f) Reticulado após o final do processo.

Figura 3.7: Evolução do AC para o cálculo das distâncias no método de [Tavakoli et al. 2008].

Para calcular o planejamento do caminho, cada célula contém além de sua distância até o objetivo, um vetor de direções ótimas para os agentes. Para calcular o vetor de direções de cada célula, deve-se fazer comparações com todas as oito células vizinhas. Para o controlador central escolher qual caminho um agente deve tomar, ele calcula um custo de movimento daquele agente de acordo com a quantidade de colisões e proximidade do objetivo. Dois caminhos possíveis para o exemplo mostrado na Figura 3.7 são mostrados na Figura 3.8. Para comprovar a eficiência do método, foi feita uma simulação com vários agentes, porém o método não foi aplicado em robôs reais.

Em 2010 [Soofiyani et al. 2010], foi publicada uma melhoria do trabalho de [Tavakoli et al. 2008]. A contribuição do trabalho é na criação de planos que preferem movimentos

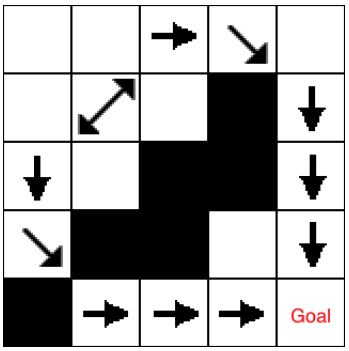


Figura 3.8: Caminhos possíveis do estado inicial até a meta pelo método de [Tavakoli et al. 2008].

em linha reta em relação a movimentos de ziguezague.

Como em [Tavakoli et al. 2008], o ambiente do problema é uma grade de células de formato quadrático, sendo cada célula representada no sistema de coordenadas cartesianas. Cada célula contém um dos três objetos: Robô, Obstáculo, Objetivo. Se uma célula contém um robô ou obstáculo, então é uma célula ocupada, caso contrário é uma célula livre.

O cálculo da distância relativa entre as células e a arquitetura do autômato celular também são iguais aquelas em [Tavakoli et al. 2008], logo o movimento em linha reta tem o custo de 10 e o movimento em diagonal tem o custo de 14, e a arquitetura do autômato celular consiste em se a célula corrente for livre, a distância é definida como o menor valor entre o seu valor corrente, os valores em linha reta mais 10 e os valores na diagonal mais 14. Aparentemente, o trabalho de [Soofiyani et al. 2010] não aceita movimentos na diagonal caso exista obstáculos acima e ao lado da célula, por exemplo, caso exista um obstáculo ao norte e um ao leste, e a célula ao sudeste seja livre, o robô não está autorizado a movimentar naquela direção, diferentemente dos trabalhos anteriores.

No final da aplicação do autômato celular, todas as células livres terão a menor distância até o objetivo. Em [Behring et al. 2000] e [Tavakoli et al. 2008], para o planejamento do caminho, o robô procura a menor distância entre seus oito vizinhos de raio um e vai para aquela direção, já neste trabalho, primeiramente o algoritmo procura o vizinho de menor distância no raio um, depois de encontrado, verifica se existe um caminho direto até o objetivo, caso exista, ele move até a posição mais próxima, depois até o objetivo, caso contrário, o algoritmo incrementa o raio e vai procurando a menor distância entre os vizinhos de raio menor, até que o menor no raio r seja maior que no raio r-1, nesse caso, o robô move para a posição de menor raio r e continua o processo. O processo pode ser visto na Figura 3.9. A comparação dos caminhos encontrados pelos métodos [Tavakoli et al. 2008] e [Soofiyani et al. 2010] são mostrados na Figura 3.10.

Assim, o robô faz uma movimentação em linha reta no maior raio possível, só então ele altera o seu movimento. Foi mostrado que o custo neste trabalho é melhor ou igual

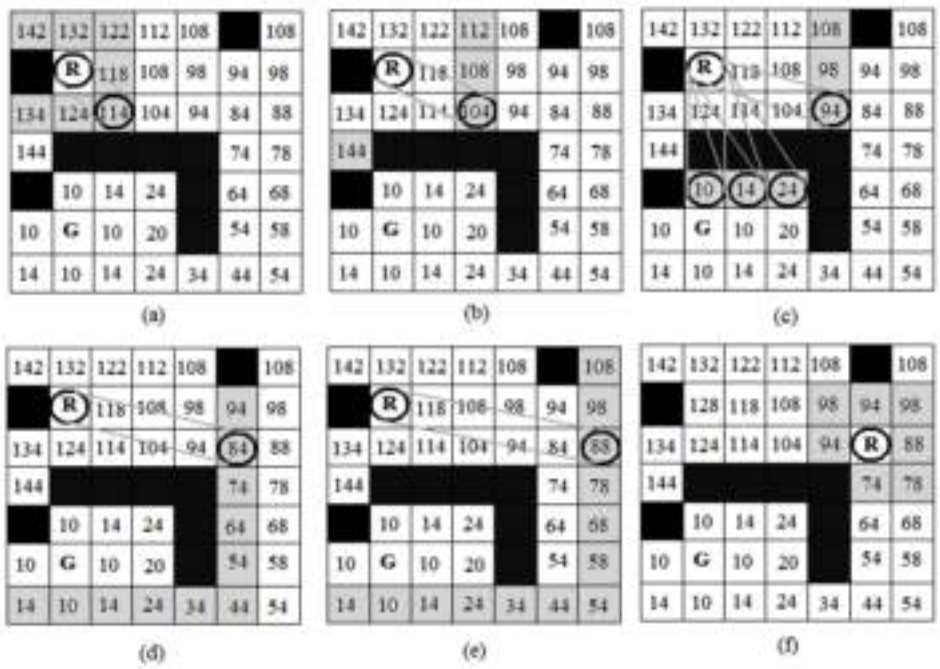


Figura 3.9: (a) A melhor célula no raio 1. (b) Melhor célula no raio 2. (c) As células com os valores 10, 14 e 24 no raio 3 não são as melhores pois contém colisões. (d)(e) Células com valores 84 no raio 4 e 88 no raio 5, são escolhidas, mas como no raio 5 o valor é maior, o robô move para a célula com valor 84. (f) Raio retorna para 1. [Soofiyani et al. 2010]

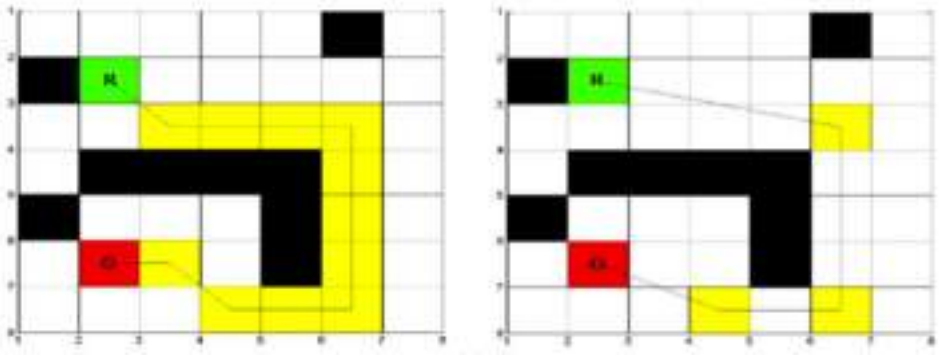


Figura 3.10: À esquerda o caminho encontrado pelo método [Tavakoli et al. 2008] e à direita o caminho encontrado pelo método [Soofiyani et al. 2010]. Para estes resultados, aparentemente foi utilizada a restrição de movimentação diagonal caso haja obstáculos ao lado e acima. [Soofiyani et al. 2010]

do que o custo em [Behring et al. 2000]. E como trabalho futuro, os autores propõem a aplicação em um ambiente dinâmico, com obstáculos e objetivos móveis. Os experimentos foram feitos em um ambiente implementado em Matlab, mas não foram utilizados robôs reais.

O último trabalho encontrado que se enquadra nesta abordagem é um pouco diferente dos demais, pois utiliza técnicas de visão computacional em auxílio ao planejamento.

Em 2012, foi publicado em [Kostavelis et al. 2012] um método que utiliza somente um agente em um ambiente que será conhecido online através da aquisição de uma imagem. O objetivo é conhecido e consiste em traçar um caminho até o horizonte da imagem adquirida.

O método utiliza um mapa de profundidade da imagem, ou seja, ao invés de utilizar a imagem pura para o planejamento do caminho, a imagem é adquirida com a informação de profundidade dos itens na cena. Apesar de não importar qual método utilizado para encontrar o mapa de profundidade, neste trabalho se utiliza a construção de um mapa de disparidade, utilizando duas imagens adquiridas por um equipamento de captura de imagens estéreo.

Após encontrar o mapa de profundidade, cria-se a chamada *v-disparity image*. Nesta imagem, cada pixel contém um valor inteiro positivo que denota o número de pixels na imagem de entrada que estão na mesma linha das ordenadas da imagem, e possuem valores de disparidade iguais as da sua abcissa. Esta informação é importante para distinguir os pixels pertencentes ao chão e aos obstáculos.

Contendo estas duas informações, ocorre um remapeamento utilizando uma transformada polar que rearranja a informação de profundidade da cena colocando os valores de disparidade em uma topologia radial em torno da frente do robô. Consequentemente, tem-se uma distribuição espacial dos obstáculos em volta do robô.

A saída da transformada são duas imagens de profundidade polar distintas. A primeira é o mapa de profundidade polar, resultado da transformação da imagem de disparidade inicial. A segunda é um mapa de profundidade sem obstáculos. Cada coluna do mapa de profundidade polar corresponde a uma direção específica na cena, ou seja, a coluna de número 90 contém a direção do robô a 90°. No fim, os dois mapas de profundidade polar são subtraídos, e é aplicado um *thresholding*, encontrando assim uma imagem binária onde os 0s correspondem ao chão e os 1s aos obstáculos.

Esta imagem binária servirá de entrada para o primeiro autômato celular que utiliza a vizinhança de von Neumann de raio um para criar o campo do chão. Este campo é um gradiente contendo valores baixos nas células próximas ao robô e valores maiores longe dele. Para obstáculos é utilizado o estado -1 e para células ainda não calculadas, o valor é 0. O autômato celular para quando todas as células estão em um estado diferente de 0, ou seja, é calculada a distância entre o robô e todas as outras células da imagem. A regra utilizada pelo autômato celular é a mesma utilizada na segunda fase de [Behring et al. 2000], mudando somente o tipo de vizinhança.

O último passo é a estimação do caminho, para isso, definiu-se que o objetivo do robô é chegar até o ponto mais distante e alcançável. Esta distância é calculada utilizando a imagem binária, mais especificamente, o objetivo está na maior linha alcançável pela imagem binária, encontrando assim o horizonte da imagem. Para o cálculo do caminho, utilizou-se outro autômato celular, agora com a vizinhança de Moore e também raio 1, e

como entrada o campo do chão gerado pelo primeiro autômato.

Iniciando na célula objetivo, percorre-se o campo do chão até chegar no mais próximo do robô, sempre passando por estados com valores menores de distância. A regra utilizada é verificar entre os vizinhos aquele que contém a menor distância, então trocar o estado dele para uma célula do caminho.

Para os testes, os autores utilizaram uma base de imagens de cenas naturais onde foi possível encontrar o caminho sem encostar nos obstáculos do ambiente. Visto que o planejamento de caminhos é feito da mesma forma do trabalho de [Behring et al. 2000], a maior contribuição do método é a criação do reticulado baseado na imagem do ambiente encontrada no momento, diferentemente dos outros métodos que assumem um reticulado plano como o ambiente do autômato celular.

3.5 Abordagem por Regra de Atualização Local [Akbarimajd e Lucas 2006, Akbarimajd e Hassanzadeh 2011, Akbarimajd e Hassanzadeh 2012, Ioannidis et al. 2008, Ioannidis et al. 2011a, Ioannidis et al. 2011b]

Essa abordagem surgiu como uma proposta de arquitetura para resolver problemas de planejamento de caminhos para robôs utilizando um método baseado em autômatos celulares [Akbarimajd e Lucas 2006]. A principal característica desta abordagem é a utilização de sensores a cada passo de tempo para avaliar a vizinhança do robô, e então utilizar a regra de transição que decide para qual célula se deve ir no próximo passo de tempo.

O método desenvolvido em [Akbarimajd e Lucas 2006], utiliza um único agente além do ambiente ser desconhecido, utilizando sensores para verificar a vizinhança da posição em que o robô se encontra no momento. O robô sabe em que posição está e também a posição do objetivo que quer chegar. Assim, a cada passo de tempo, o robô avalia sua vizinhança de Moore com três estados possíveis para cada célula: Robô, Livre e Obstáculo. Como regra de transição, o robô é trocado com algum de seus vizinhos. A escolha deste vizinho depende da posição do objetivo. Para realizar a troca a célula central que estava no estado Robô é alterada para a célula Livre e a vizinha que estava no estado Livre é alterado para o estado Robô. Inicialmente, a viabilidade do método foi mostrada através de simulações.

A arquitetura de execução do planejamento de caminhos para o robô, denominada *Saphira*, é baseada em camadas. No centro está a chamada LPS, que é a representação geométrica do espaço em torno do robô. Ela recebe dados dos sensores, além de dados de

um reconhecedor de "fim da linha". Tendo estes dados, o método decidirá através de dois tipos de ações o que deve ser feito, ações de alto-nível e de baixo-nível. As de alto-nível são aquelas orientadas ao objetivo, ou seja, escolha do conjunto de regras correspondente. Por exemplo, se o robô está em uma posição $(x,\ y)$, e o objetivo está em uma posição $(xg,\ yg)$, tal que $x < xg$ e $y < yg$, o conjunto de regras escolhido é o mostrado na Figura 3.11. As de baixo-nível são aquelas relativas ao desvio de obstáculo, ou seja, escolher qual regra dentro do conjunto escolhido previamente, deverá ser utilizada, atualizando assim a célula do autômato celular. Para o conjunto de regras R1 da Figura 3.11, caso não haja obstáculo na posição $x + 1\ y + 1$, move-se o robô para esta posição, caso contrário, tenta-se a próxima regra que verifica se a posição $x + 1\ y$ contém obstáculo, assim sucessivamente. Além disto, existe um servidor de robô que controla os dados como a operação dos robôs, o trabalho dos sensores e controle do tempo de disparo do sonar e do infravermelho, entre outros. Após as simulações, a arquitetura foi embarcada em um robô móvel com rodas, porém não especificado, se mostrando viável e apropriada.

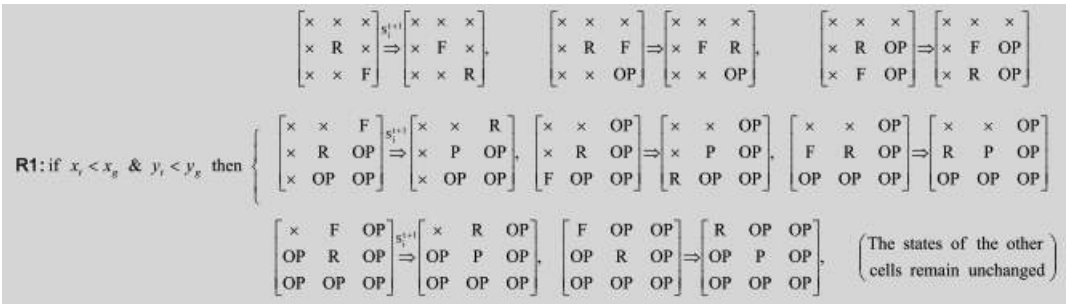


Figura 3.11: Conjunto de regras R1 correspondente à $x < xg$ e $y < yg$. R é a célula Robô, F é uma célula livre, OP é uma célula com obstáculo e x é uma célula indiferente. [Akbarimajd e Hassanzadeh 2012]

Em 2011, foi proposto um novo método para planejamento de caminhos em tempo real utilizando os autômatos celulares [Akbarimajd e Hassanzadeh 2011]. Neste trabalho, é possível tratar vários robôs e o cálculo do melhor objetivo direto é feito durante o planejamento. O autômato celular utilizado contém quatro estados: Célula Robô, Célula Livre, Célula Obstáculo e Célula com melhor objetivo direto. A cada passo de tempo, a célula robô é trocada com a célula com melhor objetivo direto, utilizando a vizinhança de Moore.

Primeiramente, o ambiente é discretizado em células do autômato celular. Um robô consegue reconhecer sua posição e se os os seus vizinhos estão livres ou se contêm algum obstáculo. Após a discretizaçào, e definição dos estados, deve-se aplicar a regra de transição no reticulado, que consiste em fazer com que o robô troque de posição com a célula com o melhor objetivo direto, deixando uma célula livre em seu lugar. A célula com melhor objetivo direto é definido como aquela célula livre, vizinha do robô, que contém a menor distância até o objetivo. Todas as outras células se mantêm inalteradas.

Esta distância não é calculada através de um autômato celular, como os da abordagem de Behring e colaboradores (Seção 3.4). Aqui, os oito vizinhos da célula em que o robô está são transformados em vetores de direção, como mostrado na Figura 3.12. Com estes vetores e o vetor contendo o objetivo, calcula-se o produto escalar entre cada um dos vizinhos e o objetivo, aquela célula com maior valor é a célula com melhor objetivo direto.

$\vec{D}_1 = [-1,-1]^T$	$\vec{D}_2 = [-1,0]^T$	$\vec{D}_3 = [-1,1]^T$
$\vec{D}_4 = [0,-1]^T$	Robot Cell	$\vec{D}_5 = [0,1]^T$
$\vec{D}_6 = [1,-1]^T$	$\vec{D}_7 = [1,0]^T$	$\vec{D}_8 = [1,1]^T$

Figura 3.12: Vetores de direção correspondentes à vizinhança da célula do robô. [Akbarimajd e Hassanzadeh 2011].

Caso se aplique o método para mais de um robô, teremos mais de um estado de Célula Robô, e a aplicação da regra de transição será feita de forma assíncrona, ou seja, atualiza-se o estado de um robô e esta mudança será vista pelo outro robô, como mostrado na Figura 3.13.

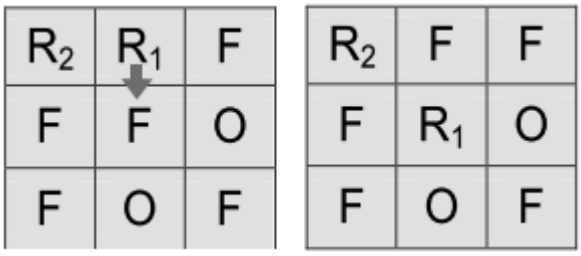


Figura 3.13: Aplicação da regra do autômato celular para o robô R1 [Akbarimajd e Hassanzadeh 2011].

O método descrito em [Akbarimajd e Hassanzadeh 2011] funciona somente para obstáculos convexos. Então, em [Akbarimajd e Hassanzadeh 2012] foi publicada uma extensão do método para aplicação em ambientes com objetos côncavos. Para isso, foi utilizado um algoritmo baseado na colônia de formigas. A ideia do algoritmo é o depósito de feromônio quando o robô está dentro de um obstáculo côncavo, assim, o autômato celular tratará células com feromônio como um obstáculo, evitando que o robô fique preso. A única mudança nas regras é a de inserção de feromônio quando encontrar três obstáculos à frente e não se movimentar para uma célula que contenha feromônio, tratando-a como um obstáculo. Como no modelo de colônia de formigas tradicional, o feromônio também é evaporado com o passar do tempo. No final, também foi implementada a arquitetura *Saphira* em um robô móvel e com rodas, e utilizando testes simulados e experimentos

em ambiente real, foi mostrado que o método é viável para ambientes com obstáculos côncavos e convexos.

Em paralelo, outro grupo também desenvolveu trabalhos seguindo a mesma abordagem de [Akbarimajd e Lucas 2006]. O primeiro trabalho deste grupo é [Ioannidis et al. 2008], que segue a ideia de planejamento de caminhos para vários robôs. A principal diferença para os trabalhos de Akbarimajd e colaboradores é a adição da restrição que os robôs devem manter determinada formação pré-definida enquanto percorrem o caminho planejado. Esta formação foi definida como uma linha reta. O algoritmo também utiliza o conceito de verificação da vizinhança a cada passo de tempo para então decidir qual a melhor posição para se movimentar. Devido à restrição proposta, caso algum dos robôs tenha que sair da formação, eles devem se comunicar para retornarem à formação o mais rápido possível.

A definição do autômato celular é similar à proposta em [Akbarimajd e Lucas 2006], pois o ambiente também é discretizado em células idênticas, e cada célula contém R+2 estados, sendo R o número de robôs. O estado 0 é uma célula livre e o estado R+1 é um obstáculo. Foi utilizada também a vizinhança de Moore e a regra de transição também verifica entre os vizinhos uma célula sem obstáculos e movimenta-se para ela.

Existem duas diferenças principais nas regras de transição para a abordagem de [Akbarimajd e Lucas 2006]. A primeira é que as regras propostas pelos autores não permitem que o robô vá para uma célula que contenha obstáculos em ambos os lados, evitando o risco de colisão. Por exemplo, caso o robô tenha que seguir para norte, e exista um obstáculo ao nordeste e outro ao sudeste, mesmo o norte sendo uma célula livre, ela não será a escolhida. A segunda diferença consiste no fato que após cada passo de tempo, existe uma segunda fase do algoritmo onde cada robô verifica se os robôs mais próximos dele estão na mesma linha e a uma distância d pré-determinada, ou seja, a formação inicial será mantida. Caso não esteja, os robôs trocam informações entre si, para determinar se devem continuar se movendo ou esperar até que os outros robôs retornem à formação.

Para verificar se o método é viável, primeiramente foi implementada uma simulação contendo 5 robôs em uma distância d de 1 célula. Os robôs encontraram o objetivo, mantendo a formação inicial. Em um segundo momento, foram utilizados 3 robôs reais do modelo e-puck [École Polytechnique Fédérale de Lausanne EPFL 2013] que contém sensores de infravermelho para detectar as células vazias vizinhas aos robôs, comunicação via Bluetooth para manter a formação e motores para a movimentação. Com isso, a cada passo de tempo, cada robô escolheu a sua regra de transição, indo para uma célula livre mais próxima do objetivo, porém mantendo a formação inicial. Este modelo será mais detalhado no Capítulo 5.

Em 2011, foram publicados dois novos artigos continuando o trabalho desenvolvido anteriormente. No primeiro artigo [Ioannidis et al. 2011a], foram mantidas as mesmas premissas do trabalho de 2008, porém agora se divide o time de robôs em subgrupos

de mesmo tamanho. Então, o método *ACO (Ant Colony Optimization)* é aplicado em cada subgrupo para diminuir a complexidade do sistema. Como cada robô move-se até o objetivo, eles criam uma trilha de feromônio e sua quantidade é detectada pelos próximos robôs, fazendo com que cada robô não necessite de comunicação com seus vizinhos a cada instante.

Existem p+2 estados possíveis para cada célula. Sendo p é o valor da célula com maior quantidade de feromônio possível, 0 a célula livre e sem feromônio, p+1 uma célula com obstáculo, e o intervalo entre r+1 e p as possíveis quantidades de feromônio.

O algoritmo funciona da seguinte forma: caso haja obstáculos a frente, aplica-se a regra para desvio de obstáculos padrão desta abordagem. Caso não haja obstáculos a frente, a próxima célula é definida utilizando a equação do ACO. Caso a formação tenha sido perdida, é aplicado o conjunto de regras de controle de formação até que se retorne à original. Para os testes do algoritmo, utilizou-se o simulador de ambiente 3d *Webots* [Cyberbotics 2013]. Todos os robôs utilizados no simulador são robôs reais que podem ser comprados. Para este trabalho foram utilizados os robôs *e-puck* modificados com trinta e seis sensores de proximidade, onde o método obteve maior precisão. Aplicando o algoritmo na simulação utilizando o Webots, verificou que o sistema encontra caminhos sem colisões, como mostrado na Figura 3.14 e concluiu-se que o método é robusto e efetivo, mesmo para ambientes com obstáculos dinâmicos.

O segundo artigo [Ioannidis et al. 2011b] acrescenta um ângulo θ relativo à orientação do robô. Assim, as regras de transição contêm o valor de θ. Outro acréscimo ao trabalho foi a utilização de mais uma formação entre os robôs, a formação triangular, além de justificar o motivo do uso de formações. Para formações em linha reta, pode-se utilizar robôs para tirar fotos panorâmicas, por exemplo. Por último, foi retirado o controle de decisão utilizado anteriormente para decidir o que um robô deve fazer no próximo passo de tempo, agora eles somente interagem entre eles para a escolha da melhor ação para manter a formação. Para os testes do algoritmo, também foram utilizadas uma simulação e testes com robôs reais. Para os robôs reais, foi feito um estudo sobre o tamanho de cada célula do reticulado deve ter para acomodar o robô e verificar se a próxima célula é livre ou não. Os autores concluem que o método é robusto e efetivo.

Figura 3.14: Time de robôs evitando colisão com um objeto retangular, os pontos são os robôs e em tons de cinza as trilhas de feromônio deixadas por eles [Ioannidis et al. 2011a].

3.6 Abordagem por Envio de Mensagens [Rosenberg 2007, Rosenberg 2008, Rosenberg 2010, Rosenberg 2012]

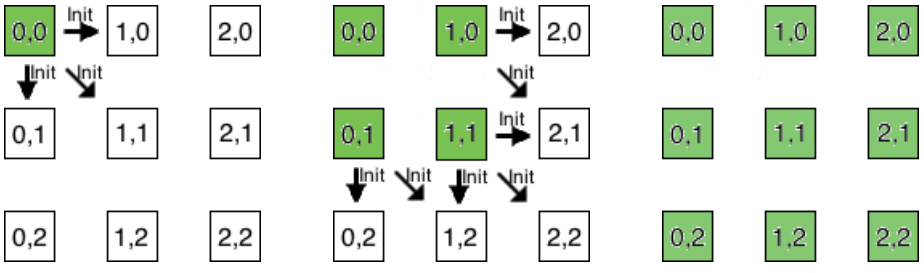
Uma abordagem encontrada na literatura objetivando o planejamento de caminhos para robôs auxiliado pelos autômatos celulares é a utilizada por Rosenberg e colaboradores. Em [Rosenberg 2007] foi publicado um modelo inspirado nas formigas onde a inteligência da busca é invertida, ou seja, ela está no ambiente e não nas formigas. As células comunicam entre as vizinhas através de mensagens contendo a existência de obstáculo, de formigas, alimento ou de feromônio. Além de mensagens de comunicação com as formigas, para execução de ações como capturar o alimento ou movimentar para determinada direção.

Neste modelo, cada robô é tratado como uma "formiga", logo podem existir várias formigas no ambiente. Elas serão capazes de andar pelo ambiente, evitando colisões com

obstáculos, e com outras formigas, e tem como objetivos encontrar alimento, e estacionarem em determinado local, mantendo uma configuração padrão para todas as formigas. A grande diferença desta abordagem para todas as outras é que todo o piso onde os robôs irão se movimentar é constituído por vários processadores simples associados às células que subdividem o piso. Os robôs não tomam nenhuma decisão sobre o próximo passo a ser dado. Essa decisão é tomada pelos processadores associados às células que formam o piso do ambiente. Portanto, os robôs somente irão receber mensagens contendo o que eles deverão fazer, e então executarão a ação ordenada.

Cada célula é capaz de detectar se existe sobre ela um obstáculo, ou uma parte dele, uma formiga, um alimento, ou ambos - formiga e alimento. Cada processador associado à célula também é capaz de se comunicar com os processadores vizinhos pela vizinhança de Moore. Essas mensagens contêm informações sobre o que cada célula vizinha tem sobre ela, além da quantidade de feromônio existente naquela posição. Por fim, cada processador também pode se comunicar com a formiga existente sobre a célula, com mensagens para pegar a comida existente, ou mover para uma posição vizinha. Tendo isso, é garantido que as formigas não colidem entre si e com os obstáculos.

O algoritmo inicia com uma mensagem de inicialização que é transmitida à partir da célula (0,0) para seus vizinhos. Todas as células que recebem uma mensagem, vão repassam às suas vizinhas na direção contrária à qual recebeu a mensagem. Por exemplo, quando a célula (1,1) recebe uma mensagem vinda do oeste, ela repassa às suas células à leste, nordeste e sudeste, ou seja, às células (2,1), (2, 0) e (2,2). Um exemplo deste processo de inicialização pode ser visto na Figura 3.15.



(a) Mensagem de inicialização transmitida à partir da célula (0,0).

(b) Células vizinhas recebem mensagem de inicialização e retransmitem.

(c) Todas as células inicializadas.

Figura 3.15: Transmissão da mensagem de inicialização pelo ambiente.

Após o processo de inicialização, dependendo do tipo de problema, temos diferentes mensagens. Para verificar a eficiência do modelo, o autor propôs dois problemas os quais o método pode resolver. O primeiro é possibilitar as formigas a estacionarem próximas a um canto e o segundo é a busca de alimento pelas formigas através do ambiente. Para o primeiro problema, basicamente as formigas vão recebendo ordens para se aproximarem ao máximo do canto, diminuindo um fator de estacionamento a cada formiga que chegar

a uma posição de estacionamento. Para o segundo problema, as células que contêm robôs e que contêm formigas disparam mensagens para suas vizinhas para que as formigas se direcionem ao alimento.

Para o problema do estacionamento, as células livres transmitem continuamente mensagens informando às suas vizinhas que estão livres, e as células que contém uma formiga transmitem às suas vizinhas uma mensagem informando que contém uma formiga. Cada célula que contém uma formiga deve a movimentar para uma direção em determinada ordem, para garantir que as células irão estacionar. Por exemplo, se deseja que as formigas estacionem à sudoeste, uma célula que contenha uma formiga tentará mover a mesma para sudoeste, caso não seja possível, tentará à sul, a terceira opção é à oeste, então noroeste e por fim sudeste. Assim foi possível fazer com que as formigas "estacionem"em algum dos cantos do ambiente.

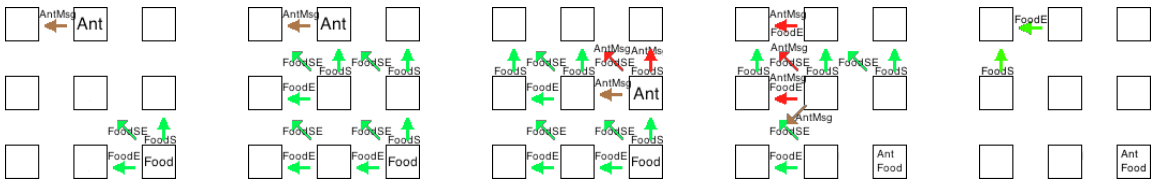
No problema das formigas procurarem alimentos, após a inicialização, as células com alimento transmitem uma mensagem aos seus vizinhos informando que contém alimento, as células com formigas transmitem em direção à célula (0,0) que contém uma formiga que necessita de alimento. Quando uma célula com formiga recebe uma mensagem de alimento, ela envia a formiga em direção de onde chegou a mensagem. Caso exista uma formiga e um alimento na mesma célula, esta envia uma mensagem para a formiga capturar o alimento e para de enviar mensagens que informando que ainda contém alimento. Porém a célula não para de retransmitir as mensagens que chegam até ela. Os autores sugerem como melhoria o envio de um valor de feromônio junto com as mensagens de alimento, este valor vai diminuindo a cada retransmissão, assim as formigas tendem a encontrar os alimentos mais próximos. O algoritmo termina quando a célula (0,0) para de receber mensagens de alimento ou mensagens de formigas sem alimento. O processo de busca por alimentos de uma formiga pode ser visualizado na Figura 3.16.

Em 2008, foi publicada [Rosenberg 2008] uma continuação deste trabalho. A principal contribuição é a utilização de dois outros problemas para a prova de conceito do modelo. Além dos dois problemas utilizados anteriormente, foram também utilizados o problema de um robô encontrar a saída de um labirinto e incrementado o problema de encontrar comida para formigas chamadas "ativas", ou seja, elas também se movimentam em busca de comida ao invés de só esperarem as mensagens de direção do alimento.

Neste trabalho também foi melhor explicado o modo de ativação das células, e foi feita uma análise mais profunda da complexidade dos algoritmos para resolverem os problemas.

Já em 2010, foi publicado em [Rosenberg 2010] uma nova forma de resolver o problema do "estacionamento de formigas", exposto pela primeira vez em [Rosenberg 2007]. Neste trabalho, a máquina de estados finita resiste na formiga e não mais no ambiente, excluindo assim a utilização dos autômatos celulares no planejamento de caminhos.

Por fim, em 2012 foi publicado em [Rosenberg 2012] outro trabalho com o mesmo título do [Rosenberg 2007]. Nele, o autor explica com mais detalhes os trabalhos anterio-



(a) Primeiro passo de tempo após a inicialização das células.

(b) Segundo passo de tempo. Célula com formiga recebe mensagem de alimento vindo do sul e do sudeste.

(c) Terceiro passo de tempo. Após a movimentação da formiga para sudeste, a célula recebe mensagem de alimento vindo do sul.

(d) Quarto passo de tempo. Após a movimentação da formiga para o sul, a célula que contém formiga e alimento, então envia uma mensagem para a formiga pegar o alimento e cessa o envio de mensagens de alimento e de formiga sem alimento.

(e) Quinto passo de tempo. Transmissão das últimas mensagens de alimento.

Figura 3.16: Transmissão da mensagem de busca de alimentos. Setas verdes são mensagens de informação de alimento, setas marrons são mensagens de informação de formiga sem alimento, setas vermelhas são o envio das duas mensagens no mesmo passo de tempo.

res, além de propor alguns trabalhos futuros, como: novas ideias para utilizar o método como modelo discreto para sistemas imunológicos naturais; melhorar a performance dos algoritmos; desvio de computadores com falhas; entre outros.

3.7 Comparação das Características das Abordagens

A Tabela 3.1 contém as características específicas de cada uma das abordagens de planejamento de caminhos para robôs móveis que utilizam autômatos celulares.

Primeiramente, a Tabela 3.1 mostra a quantidade de trabalhos existentes em cada abordagem. A abordagem por Camadas e Atração para o Objetivo e a abordagem por Envio de Mensagens contém 4 trabalhos, porém todos são do mesmo grupo de pesquisa. A abordagem por Difusão de Distância à Meta contém 4 trabalhos de diferentes grupos de pesquisa, sendo que o primeiro trabalho da abordagem é bastante citado como exemplo de planejamento de caminhos utilizando autômatos celulares, devido à isso, além de ter a implementação bastante simples, este foi o primeiro trabalho investigado, como será mostrado no Capítulo 4.

Especificamente sobre a característica dos autômatos celulares, a Tabela 3.1 mostra que ambas as vizinhanças, von Neumann e Moore, foram utilizadas, sendo que a grande maioria dos trabalhos utilizam as vizinhanças com o raio 1. Existe também uma distinção na quantidade de autômatos celulares exigidas em cada abordagem. Na abordagem por Camadas e Atração para o Objetivo, temos 5 autômatos celulares onde cada um representa uma camada do modelo. Já nos trabalhos da Abordagem por Regra de Atualização Local, podem existir até 3 ACs que serão evoluídos dependendo dos dados lidos pelos sensores. As regras relativas aos ACs de cada abordagem também podem ser vistos na Tabela 3.1. Destacam-se as regras de controle de formação do trabalho de Ioannidis e colaboradores, que foram relevantes para a escolha do trabalho para implementação.

A linha Agentes mostram quantos robôs existiam no ambiente nos trabalhos de cada abordagem, sendo que com exceção dos trabalhos da Difusão da Força e do Diagrama de Voronoi, todos possibilitam a existência de mais de um robô no ambiente. O ambiente conhecido é uma exigência de todas as abordagens, com exceção da abordagem de Regra de Atualização Local, que utiliza os sensores dos robôs a cada passo de tempo, para construir o ambiente o qual os robôs irão se movimentar. Esta característica foi relevante para a escolha de um trabalho desta abordagem para investigação, que será mostrada no Capítulo 5.

Quase todos os trabalhos contém uma Meta bem definida, de onde o robô saindo de uma posição inicial deve chegar ao final do processo. A abordagem do Diagrama de Voronoi não contém uma meta definida pois o robô deve passar por todos os caminhos o mais distante possível dos obstáculos. Portanto, os Objetivos dos trabalhos são em sua grande maioria encontrar um caminho sem colisões até a meta, porém como dito anteriormente, a abordagem do Diagrama de Voronoi deseja que o robô passe o mais distante de todos os obstáculos e os trabalhos da abordagem por Envio de Mensagens tem mais dois objetivos que são o Estacionamento de robôs e a resolução de labirintos.

Alguns trabalhos também impõem algumas restrições. Os trabalhos da abordagem de Camadas e Atração para o Objetivo tem como restrição que os robôs só podem visitar as células com um ângulo máximo de rotação, não existindo a rotação sobre o próprio eixo. Na abordagem do Diagrama de Voronoi, foram utilizados robôs em formato de diamante. Nos trabalhos de Ioannidis e colaboradores, que fazem parte da abordagem por Regra de Atualização Local, a formação dos robôs deve ser mantida o maior tempo possível e quando desmanchada, deve ser retomada o mais rápido o possível. Por fim, a abordagem por Envio de Mensagens, exige que "computadores" existam no piso e que possam comunicar entre si e com os agentes existentes no ambiente.

Além dos ACs, algumas abordagens contém trabalhos que investigam a utilização de outros modelos inteligentes. A abordagem por Regra de Atualização Local, contém dois trabalhos, de autores diferentes, que utilizam o algoritmo de colônia de formigas de diferentes formas. Já a abordagem por Envio de Mensagens utiliza o conceito de formigas

no ambiente, porém sem utilizar o modelo original baseado na colônia de formigas.

Por fim, a única abordagem onde foram apresentados experimentos com robôs reais foram da abordagem por Regra de Atualização Local. Os trabalhos de Ioannidis e colaboradores mostram experimentos com os robôs e-pucks, os mesmos que existem no laboratório de computação Bio-Inspirada da Universidade Federal de Uberlândia. Esta característica também foi importante para a escolha dos trabalhos de Ioannidis e colaboradores para investigação.

Abordagem	**Difusão de Força**	**Camadas e Atração para o Objetivo**	**Diagrama de Voronoi**	**Difusão da Distância à Meta**	**Regra de Atualização Local**	**Envio de Mensagens**
Quantidade de Trabalhos	1	4	1	4	6	4
Referências	[Shu e Buxton 1995]	[Marchese 1996, Marchese 2002, Marchese 2005, Marchese 2011]	[Tzionas et al. 1997]	[Behring et al. 2000, Tavakoli et al. 2008, Soofiyani et al. 2010, Kostavelis et al. 2012]	[Akbarimajd e Lucas 2006, Akbarimajd e Hassanzadeh 2011, Akbarimajd e Hassanzadeh 2012, Ioannidis et al. 2008, Ioannidis et al. 2011a, Ioannidis et al. 2011b]	[Rosenberg 2007, Rosenberg 2008, Rosenberg 2010, Rosenberg 2012]
Vizinhança	von Neumann	Moore	von Neumann	Moore	Moore	Moore
Quantidade de ACs	1	5	2	2	1, 2 ou 3	1
Regras	* Baseadas na força de difusão	* Bordas * Objetivo * Posição Inicial * Atração ao Objetivo * Extração do Caminho	* Detecção de Bordas * Criação do Diagrama de Voronoi	* Crescimento das Bordas * Cálculo da Distância	* Troca estado do robô pela célula livre mais próxima do objetivo * Regras de controle de formação nos trabalhos de Ioannidis	* Células enviam mensagens sobre seu estado para as suas vizinhas
Agentes	Um	Um / Multi	Um	Um / Multi	Multi	Multi
Ambiente	Conhecido	Conhecido	Conhecido	Conhecido	Desconhecido	Conhecido
Meta	Definida	Definida	-	Definida	Definida	Definida
Objetivos	Caminho até a meta	Caminho até a meta	Maior distância entre os obstáculos	Caminho até a meta	Caminho até a meta	Estacionamento de robôs, busca de alimentos, resolução de labirintos
Restrições	-	Robôs com ângulo máximo de movimentação	Robôs com formato de diamante	-	Trabalho de Ioannidis et al. mantém a formação de robôs	Computadores no piso do ambiente
Modelos Extras	Nenhum	Nenhum	Nenhum	Nenhum	Pode usar ACO	Conceito de formigas mas não utiliza o modelo
Tipos de Experimentos	Simulação	Simulação	Simulação	Simulação	Simulação e Robôs Reais	Simulação

Tabela 3.1: Tabela com o resumo das abordagens de planejamento de caminhos para robôs utilizando autômatos celulares.

Capítulo 4

Investigação sobre a Abordagem por Difusão da Distância à Meta

Após a pesquisa sobre modelos que utilizam ACs no planejamento de caminhos e a análise comparativa sintetizada na Tabela 3.1, decidimos selecionar alguns desses modelos para implementação e uma investigação mais aprofundada. Foi decidido que o primeiro trabalho a ser reproduzido seria da Abordagem por Difusão da Distância à Meta, mais especificamente o primeiro trabalho desta abordagem [Behring et al. 2000]. Este capítulo mostra de forma detalhada a implementação deste modelo, que foi escolhido por dois motivos principais, sendo o primeiro pela implementação bastante simples, possibilitando aprender a implementar os conceitos ligados à robótica e, mais especificamente aos simuladores, sem se preocupar tanto com a dificuldade do modelo específico de planejamento. O segundo motivo relevante foi devido à quantidade de trabalhos que citam o trabalho de Behring e colaboradores como exemplo de planejamento de caminhos para robôs que utilizam autômatos celulares, além de três grupos distintos terem feito trabalhos que tiveram como base o modelo de Behring e colaboradores.

Este capítulo está organizado da seguinte forma: na próxima seção, embora tenha sido descrito anteriormente, o modelo [Behring et al. 2000] será descrito detalhadamente, porém de uma forma mais próxima de como foi implementado nos experimentos, visando uma melhor explicação. Entretanto, cabe salientar que o resultado final é o mesmo. Depois é apresentada uma análise do modelo original a partir dos resultados dos testes em simulação. Posteriormente, será descrito o novo modelo baseado na abordagem Difusão da Distância à Meta, que foi proposto para corrigir os problemas encontrados em simulação. Ao final do capítulo, serão mostradas as conclusões obtidas após a investigação do modelo.

4.1 Modelo de Behring e colaboradores (2000)

Essa abordagem é simples e bastante citada como exemplo de planejamento de caminhos utilizando autômatos celulares [Parker et al. 2003], [Cai et al. 2007], [Schmidt e Fey 2010]. Nela, o planejamento consiste no cálculo da distância entre o ponto inicial e a meta, tendo o auxílio de um AC.

Em [Behring et al. 2000], o robô é considerado um único ponto sem orientação e sem aplicação das leis da dinâmica e cinemática. Consequentemente, um movimento de um passo para determinada direção acontece sempre da forma perfeita. O ambiente é definido como um espaço Euclidiano de duas dimensões e discretizado em células quadradas de forma que o robô seja do tamanho de uma célula.

Em uma fase de pré-processamento, o robô receberá como entrada uma imagem contendo a posição de cada obstáculo no ambiente e transformará em um espaço celular com quatro possíveis estados: Livre, Obstáculo, Posição Inicial e Objetivo.

O algoritmo é dividido, em três fases, sendo que as duas primeiras são executadas através da aplicação de autômatos celulares. Na primeira, ocorre o crescimento das bordas dos obstáculos, para que de certa forma se compense o fato do modelo não considerar as leis da dinâmica e cinemática, e o robô não colida com os obstáculos. O segundo autômato celular realiza o cálculo das distâncias das células até o objetivo. Os ACs utilizados nas duas fases utilizam a vizinhança de Moore. A última fase iniciará com as distâncias calculadas e planejará um caminho entre o ponto inicial e a meta.

Basicamente, a regra do autômato celular para o crescimento das bordas dos obstáculos é: se uma célula estiver no estado Livre e algum dos seus vizinhos for um obstáculo, ela se transforma em Obstáculo; caso contrário, mantém o seu estado no último passo de tempo. Esta regra será aplicada por uma quantidade fixa de iterações (x), que dependerá da resolução do espaço celular e do tamanho do robô. Em [Behring et al. 2000], foram utilizadas 4 iterações, o que resulta no alargamento dos obstáculos em uma espessura de 4 células ($x = 4$). O comportamento deste AC à partir de um único obstáculo quadrado pode ser visto na Figura 4.1, considerando $x = 2$.

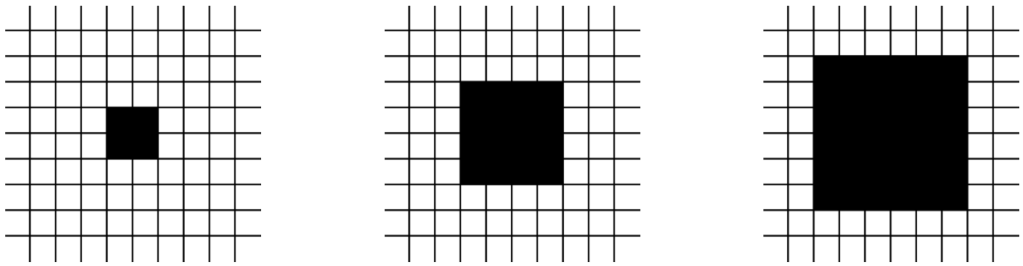


Figura 4.1: Crescimento das bordas por dois passos de tempo ($x = 2$).

Na segunda fase do algoritmo, o qual é realizado o cálculo das distâncias, o reticulado final obtido após o crescimento das bordas é utilizado como entrada. Neste autômato,

o estado de cada célula é composto por um par de subestados: (*s1*, *s2*): *s1* é o sub-estado que contém a mesma informação da primeira fase, sendo possível ocorrer um dos 4 valores - Livre, Obstáculo, Posição Inicial e Objetivo; *s2* é o sub-estado que contém o valor de distância entre a célula e o objetivo. No instante inicial tem-se: (i) todas as células livres e a célula com o robô (*s1*=Livre ou *s1*= Posição Inicial) do reticulado iniciam com o valor de *s2* indefinido; (ii) a célula objetivo (*s1* = Objetivo) recebe o valor 0; (iii) todas as células que representam obstáculos (original ou obstáculo alargado, ou seja, *s1* = Obstáculo) também têm seu valor indefinido. A regra de transição, que altera apenas o valor de *s2*, é aplicada apenas nas células com estado *s1* diferente de obstáculos. Ou seja, as células com *s1* = Obstáculo permanecem com o valor de *s2* indeterminado após a aplicação da regra. A regra de transição determina que, caso o estado *s1* da célula seja Livre, e exista algum vizinho que já teve o valor de *s2* calculado, o valor de *s2* da presente célula é alterado para o valor do vizinho mais um. Caso contrário, mantém o valor do sub-estado *s2* da célula. Após a aplicação da regra por sucessivos passos, o AC faz com que a distância seja calculada a partir do objetivo até encontrar a posição inicial, através do cálculo do valor de *s2* para as células livres, ou então até não encontrar mais células livres. Esta última situação ocorre caso não haja caminho sem colisões entre a posição inicial e o objetivo. A Figura 4.2 apresenta um exemplo de cálculo do valor de *s2* das células livres passo-a-passo.

Na etapa final, o caminho entre a posição inicial do robô e o objetivo é planejado utilizando os sub-estados *s2* das células livres. O caminho planejado pelo robô será qualquer rota que saia da posição inicial até o objetivo, sempre diminuindo em 1 o valor de sua distância, dado pelo valor de *s2*. A Figura 4.3 apresenta o caminho final calculado para o exemplo da Figura anterior.

Os experimentos em [Behring et al. 2000] foram feitos utilizando um robô real em uma mesa de teste baseada nas especificações da RoboCup, além de uma câmera fotográfica que registrava fotos do ambiente para que o caminho fosse planejado. Porém, no trabalho só foi mostrado que o robô encontra o caminho até a meta, não mostrando de fato a movimentação do robô seguindo as leis da dinâmica.

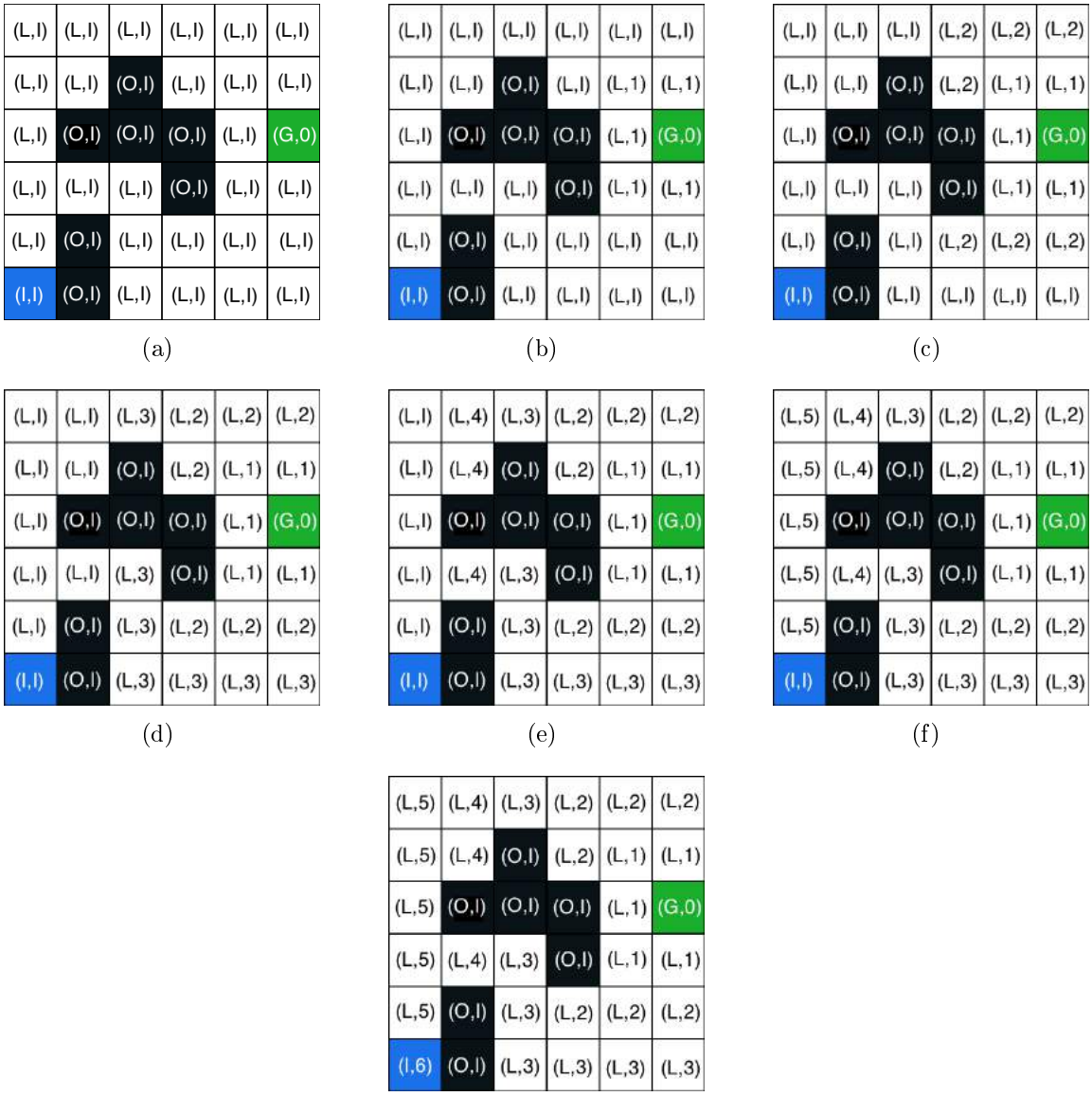


Figura 4.2: Cada célula contém o par (*s1*,*s2*), sendo que os valores de *s1* consistem em *O* para obstáculo, *L* para célula livre, *I* para a célula inicial e *G* para a célula objetivo. Os valores possíveis para *s2* são *I* para indeterminado ou um número natural caso o seu valor já tenha sido calculado.

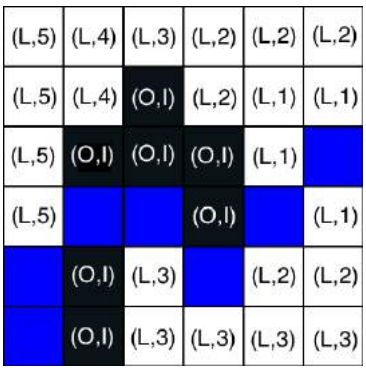


Figura 4.3: Caminho planejado pelo algoritmo.

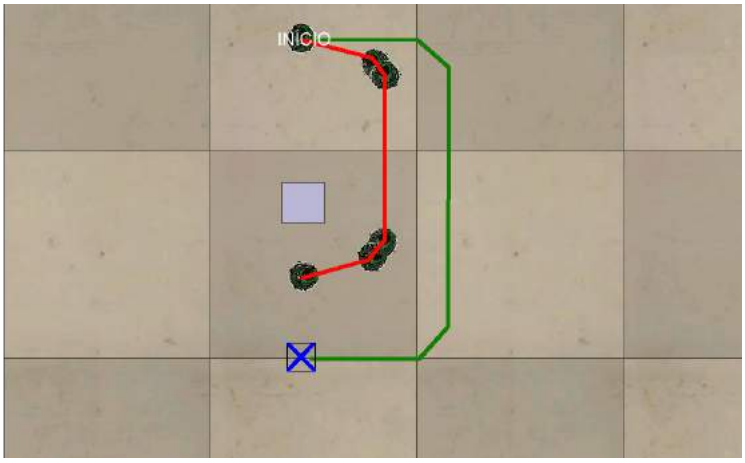
4.2 Análise do Modelo Original

Em uma primeira etapa desse trabalho, o modelo descrito por Behring e colaboradores em [Behring et al. 2000] foi implementado em um ambiente simulado, utilizando o software V-Rep [Robotics 2013]. Nessa simulação, a arquitetura adotada simula os robôs e-pucks [École Polytechnique Fédérale de Lausanne EPFL 2013]. A partir dos resultados de simulação, foram identificados dois problemas críticos no modelo. O primeiro problema refere-se ao fato da posição final alcançada pelo robô nas simulações ser distante da meta desejada, em diversos cenários avaliados. Um exemplo dessa situação é apresentado na Figura 4.4.a, que indica a rota calculada pelo modelo no início da execução, e a rota de fato percorrida pelo robô e-puck na simulação. Essa diferença entre o planejado e o realizado ocorre em razão do modelo ignorar a dinâmica do robô durante o percurso do caminho calculado. Essa dinâmica é considerada pelo ambiente de simulação fazendo com que um passo em determinada direção não seja realizado de forma perfeita. Cabe destacar também na simulação ilustrada na Figura 4.4.a que o robô alcança uma posição final diferente da meta desejada. De forma geral, quanto maior for o número de passos de deslocamento e giros que o robô precisa efetuar para realizar a rota planejada, maior a tendência da rota executada divergir da planejada e a posição final ser distante da meta. Cabe salientar que parte do problema relativo à posição final do robô ser distante da meta é devido à forma como foi implementada a movimentação do robô, sem a utilização da odometria, como explicado na Seção 2.2.3.

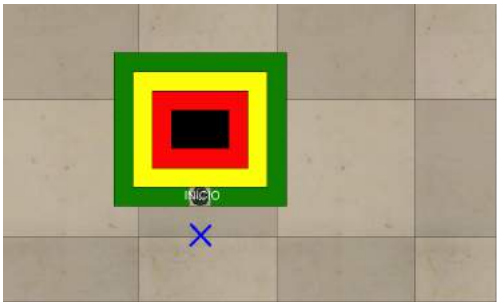
O segundo problema ocorre quando o robô tem sua posição inicial dentro da área de um obstáculo que foi aumentado. Nesse caso, o modelo é incapaz de encontrar um caminho até a meta, mesmo este existindo no ambiente original. A Figura 4.4.b ilustra uma situação em que isso acontece. Nessa figura, os obstáculos originais são representados por células na cor preta, as áreas alargadas após a aplicação do AC na primeira fase do algoritmo são representadas em vermelho (x=1), em amarelo (x=2) e em verde (x=3). As demais células (livres) estão representadas em cinza. Nesse exemplo, vemos que o robô se encontra em uma célula relativamente próxima da meta, mas em uma área de alargamento do obstáculo original (x=3). Caso esse seja o cenário inicial ao se aplicar o algoritmo descrito em [Behring et al. 2000], o modelo não é capaz de planejar uma trajetória para o robô, uma vez que o cálculo das distâncias não atinge as células alargadas.

Uma variação dessa situação ocorre quando o robô encontra-se inicialmente em uma posição que embora não pertença a área de algum obstáculo aumentado, está circundado por áreas desse tipo impedindo que o modelo encontre um caminho para a meta. A Figura 4.4.c ilustra essa situação. Nela, o robô está inicialmente em uma célula cinza, ou seja, livre. Entretanto a região cinza onde o robô se encontra está circundada por regiões alargadas de obstáculos, o que impede que o algoritmo original descrito em [Behring et al. 2000] encontre uma rota para o robô até a meta. A solução proposta para estes problemas

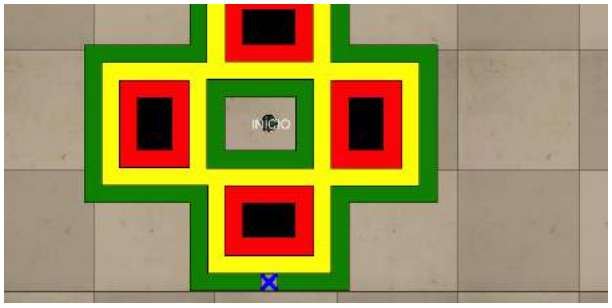
será apresentada na próxima seção, a partir das modificações propostas no modelo.



(a) Diferença entre a rota planejada (em verde) e a rota percorrida (em vermelho).



(b) Robô se encontra próximo de um obstáculo que cresceu.



(c) Robô se encontra em uma célula livre, porém que está cercada de obstáculos.

Figura 4.4: Exemplos de problemas observados quando implementamos o modelo original de Behring e colegas [Behring et al. 2000] no ambiente de simulação V-REP.

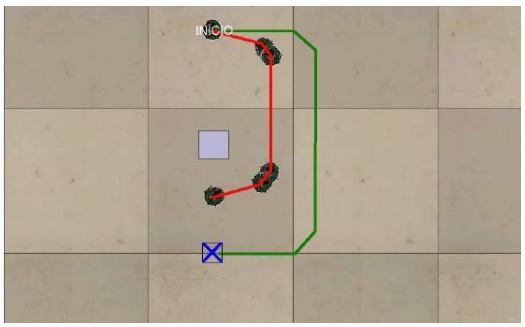
4.3 Novo Modelo Baseado na Difusão da Distância

4.3.1 Primeira Adaptação: Recálculo da rota a cada n passos

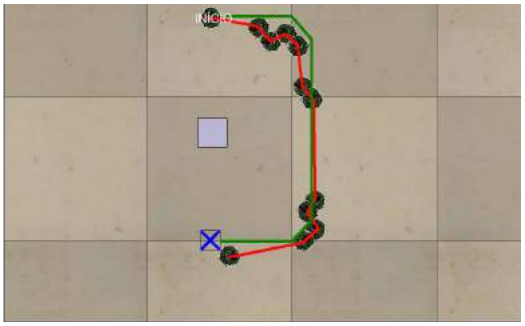
O primeiro problema foi identificado quando o modelo original foi implementado em um ambiente simulado com dinâmica. A solução adotada tem como objetivo aproximar a rota planejada do caminho efetivamente percorrido pelo robô. Para tal, o modelo foi alterado de forma que o algoritmo realiza a difusão das distâncias e a geração de um novo caminho até a meta a cada n passos. Assim, mesmo que o robô esteja em uma posição diferente da que era planejada após se movimentar por uma sequência de passos, um novo plano será gerado a partir de sua nova posição até a meta. Como consequência dessa alteração, a posição final do robô se aproxima da meta desejada.

Durante os experimentos de simulação realizados, foi utilizado um valor de n igual a 5, obtendo assim uma maior aproximação entre a rota planejada e a executada ao final

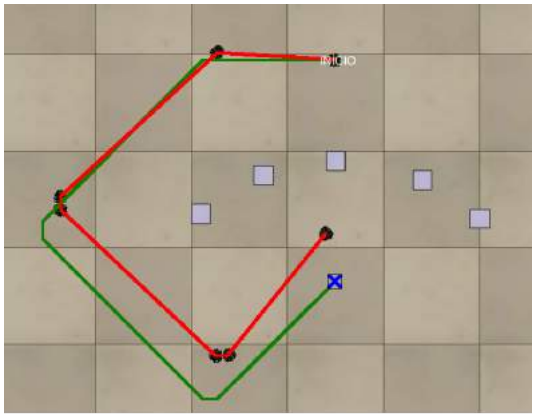
do processo, além de ser possível atingir a meta. A Figura 4.5 ilustra alguns exemplos com a rota inicial planejada pelo algoritmo e a rota executada pelo e-puck no ambiente de simulação. Para cada cenário, foi apresentado o comportamento do robô no modelo original (que realiza o cálculo da rota uma única vez) e no modelo adaptado (que realiza o cálculo a cada n vezes).



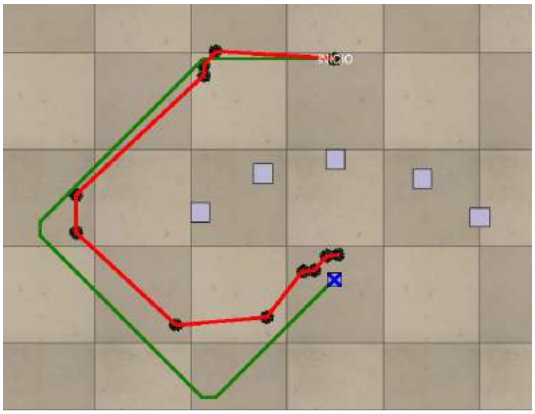
(a) Modelo original com um obstáculo.



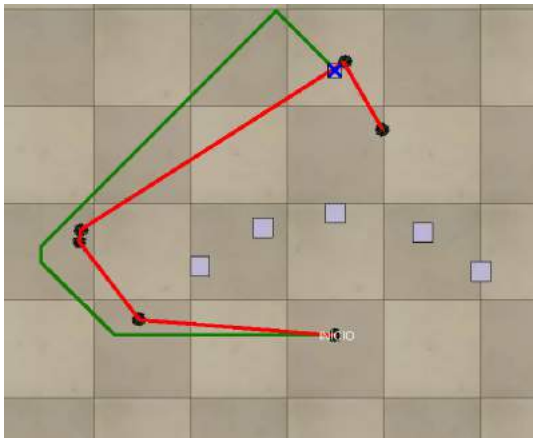
(b) Modelo modificado com um obstáculo.



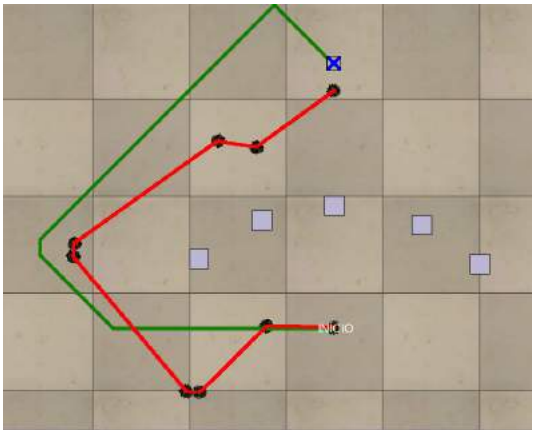
(c) Modelo original com cinco obstáculos ao sul.



(d) Modelo modificado com cinco obstáculos ao sul.



(e) Modelo original com cinco obstáculos ao norte.



(f) Modelo modificado com cinco obstáculos ao norte.

Figura 4.5: Percursos encontrados pelos modelos. Em azul, a meta, em verde o caminho planejado no início do processo e em vermelho o caminho percorrido pelo robô.

Para a efetiva utilização em dispositivos reais, tanto do modelo original quanto do

modelo adaptado, é necessário que o algoritmo seja capaz de receber uma imagem do ambiente a ser percorrido, com a devida identificação dos obstáculos. No caso do modelo adaptado, essa captura da imagem e processamento dos obstáculos deve ser refeita antes de cada recálculo das distâncias e do caminho até a meta. Dessa forma, o valor do n mais adequado pode ser definido em função da latência provocada pelo sistema de processamento dessas imagens.

Dessa forma, para uma aplicação do modelo em robôs reais, é necessário implementar um sistema de captura e processamento de imagens a ser acoplado ao sistema. Este sistema não foi implementado nessa dissertação por fugir do escopo do trabalho, portanto a implementação de tal sistema é sugerida para trabalhos futuros em um projeto multi-disciplinar com a inclusão da área de processamento de imagens.

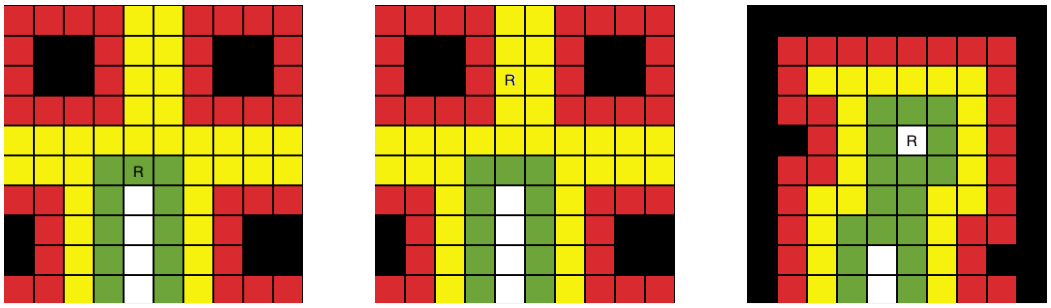
4.3.2 Segunda Adaptação: Difusão da Distância na Área dos Obstáculos Alargados

O segundo problema identificado no modelo original [Behring et al. 2000] ocorre quando o algoritmo não consegue encontrar um caminho até a meta caso o robô se encontre inicialmente dentro da área alargada dos obstáculos, ou mesmo se o robô se encontra em uma célula livre fora do alargamento dos obstáculos mas a única alternativa para caminhar até a meta o faria passar por uma área desse tipo. Embora uma situação assim possa ocorrer desde a posição inicial quando se aplica o cálculo das distâncias pela primeira vez, ela se torna mais crítica quando a primeira adaptação descrita na seção anterior é aplicada. Ou seja, quando se aplica o reprocessamento das distâncias a cada n passos, aumenta-se a probabilidade desse tipo de situação ocorrer, pois devido as restrições na implementação de sua dinâmica o robô provavelmente não irá percorrer a rota exatamente como inicialmente planejado e pode ser que tenha invadido uma área de alargamento de obstáculos, no momento em que o cálculo da rota é refeito. Essa situação ocorreu com frequência nos experimentos simulados.

A Figura 4.6 ilustra 3 cenários nos quais uma situação desse tipo foi identificada, sendo que em todas foi aplicado um fator de alargamento igual a três (x=3). Nessa figura, as células em preto indicam a posição real dos obstáculos, as células em vermelho indicam as áreas alargadas mais próximas dos obstáculos (x=1), as células em amarelo indicam as áreas alargadas que distam 2 células dos obstáculos reais (x=2) e as células em verde representam as áreas com x=3. As células livres fora da região de alargamento, estão representadas em branco. Nas Figuras 4.6.a e 4.6.b o robô encontra-se inicialmente em uma região imprópria ao calculo da rota no modelo original, após o alargamento dos obstáculos. Na Figura 4.6.a o robô encontra-se em uma célula verde (ou seja, distante em 3 células do obstáculo original), enquanto na Figura 4.6.b o robô se encontra em uma célula amarela (x=2). O motivo do modelo original não conseguir calcular um caminho

do robô ate a meta reside no simples fato do algoritmo tratar todas as células (pretas, vermelhas, amarelas e verdes) como obstáculos reais, não realizando o cálculo da distância para essas células. Dessa forma, não é possível encontrar um caminho da posição inicial do robô ate a meta.

A Figura 4.6.c ilustra uma situação um pouco diferente das duas anteriores. Nesse caso, o robô está em uma célula livre. Entretanto, a área onde o robô se encontra é circundada por áreas de alargamento de obstáculos, formando uma espécie de "armadilha" de onde o robô não consegue sair. Nesse caso, embora o robô esteja em uma célula livre, o algoritmo não consegue realizar o cálculo das distâncias até a célula inicial porque as células na área alargada formam uma espécie de barreira para essa difusão. Nesse terceiro caso, o algoritmo também não é capaz de encontrar uma rota que leve o robô ate a meta, embora ela exista.



(a) O robô está em uma distância de três células de um obstáculo real.

(b) O robô está em uma distância de duas células de um obstáculo real.

(c) O robô está em uma célula livre, mas está em uma "armadilha" de obstáculos.

Figura 4.6: Exemplos de quando o modelo original não consegue encontrar um caminho até a meta.

Para a solução dos problemas ilustrados na Figura 4.6, foram utilizadas duas medidas que passam a ser calculadas nas células que se tornaram áreas de alargamento de obstáculos, após a aplicação da primeira fase do algoritmo. Essas medidas foram integradas como sub-estados dessas células. A primeira irá medir a menor distância da célula que está em uma área de alargamento até uma célula livre, semelhantemente ao modelo original, porém ao invés de buscar o menor caminho até a meta, busca-se o menor caminho até uma célula livre. Essa medida será utilizada posteriormente na construção da rota final pelo algoritmo partindo-se do princípio que se um robô está numa área de alargamento de um obstáculo, o algoritmo deve primeiramente escolher um caminho que tire o robô o mais rápido o possível da proximidade de um obstáculo para depois iniciar um caminho em que se preocupe em levar o robô rapidamente à meta. Entretanto, aliado a essa ideia, utilizamos uma segunda medida para se verificar um segundo princípio que nessa trajetória de saída de uma área alargada, o robô deve passar o mais distante o possível de um obstáculo real. Dessa forma, a segunda medida é utilizada para medir o quão próximo cada célula das regiões alargadas está de um obstáculo real (similar às cores utilizadas na

Figura 4.6).

Essas medidas são calculadas através de dois novos ACs, que são utilizados em uma terceira nova fase do algoritmo, que é executada caso ao final da segunda fase seja verificado que a difusão da distancia não chegou até a célula com a posição atual do robô. Caso essa situação seja identificada no final da segunda fase, conclui-se que o robô está dentro de uma área de alargamento de algum obstáculo, ou existe uma barreira de áreas alargadas de obstáculos que impediu a difusão da distância até a posição inicial do robô. Dessa forma, a terceira fase calcula novas métricas para que posteriormente o algoritmo tente achar uma rota que primeiramente tire o robô da região atual (seguindo os dois princípios de tentar fazer o robô chegar em uma célula livre o mais rápido o possível e passando o mais distante o possível de obstáculos reais), para posteriormente encontrar um caminho mais curto até a meta.

Finalmente, a quarta fase do modelo novo é similar a última fase do modelo original, ou seja, é a fase responsável por achar a rota final entre o robô e a meta, após as fases de difusão das distâncias (fase 2 e fase 3, quando necessária). A diferença ocorre quando a fase 3 foi necessária, pois o cálculo da rota final levará em conta o fato do robô estar posicionado inicialmente dentro (ou próximo) de áreas alargadas que impedem a geração de uma rota como no modelo original.

Para que essa adaptação fosse incorporada ao novo modelo, a primeira modificação se refere a representação dos estados de cada célula. No novo modelo, o estado de cada célula passa a ser representado por uma tripla de sub-estados *(s1, s2, s3)*. O sub-estado *s1* permanece similar ao utilizado no modelo anterior, apenas que agora existe um valor diferenciado para obstáculos reais e obstáculos alargados. ou seja, *s1* é o sub-estado que contém a mesma informação da primeira fase, sendo possível ocorrer um dos 5 valores: Livre, Obstáculo-real, Obstáculo-alargado, Posição Inicial e Objetivo. O sub-estado *s2* passa a ter duas funções diferentes, dependendo da natureza da célula (valor de *s1*). Para células livres ou na posição inicial do robô, ou mesmo meta (*s1*=Livre ou *s1*= Posição Inicial ou *s1*=Objetivo), o sub-estado *s2* contém o valor de distância entre a célula livre e o objetivo, que é calculado na fase 2 para esses tipos de células. Entretanto, ao final da fase 2, no modelo original em [Behring et al. 2000], as células com *s1*=Obstáculo permanecem com os valores de *s2* indefinidos. No novo modelo, após a fase 2, caso a fase 3 seja executada (após identificar que a célula que contém *s1* = Posição Inicial permanece com seu valor de *s2* indeterminado), as células com *s1*=Obstáculo-alargado terão o valor de *s2* calculado, sendo que nesse caso ele armazenará o valor da menor distância de uma célula obstáculo até uma célula livre. O sub-estado *s3* irá armazenar o valor da distância até um obstáculo real para todas as células que sejam obstáculos (*s1*=Obstáculo-real ou *s1*=Obstáculo-alargado). Para todos os outros tipos de célula (*s1*=Livre ou *s1*= Posição Inicial ou *s1*=Objetivo) o valor de *s3* é igual a zero. Os valores de *s2* e *s3* na terceira nova fase do algoritmo serão dados por dois novos ACs cujo funcionamento das regras é

explicado a seguir:

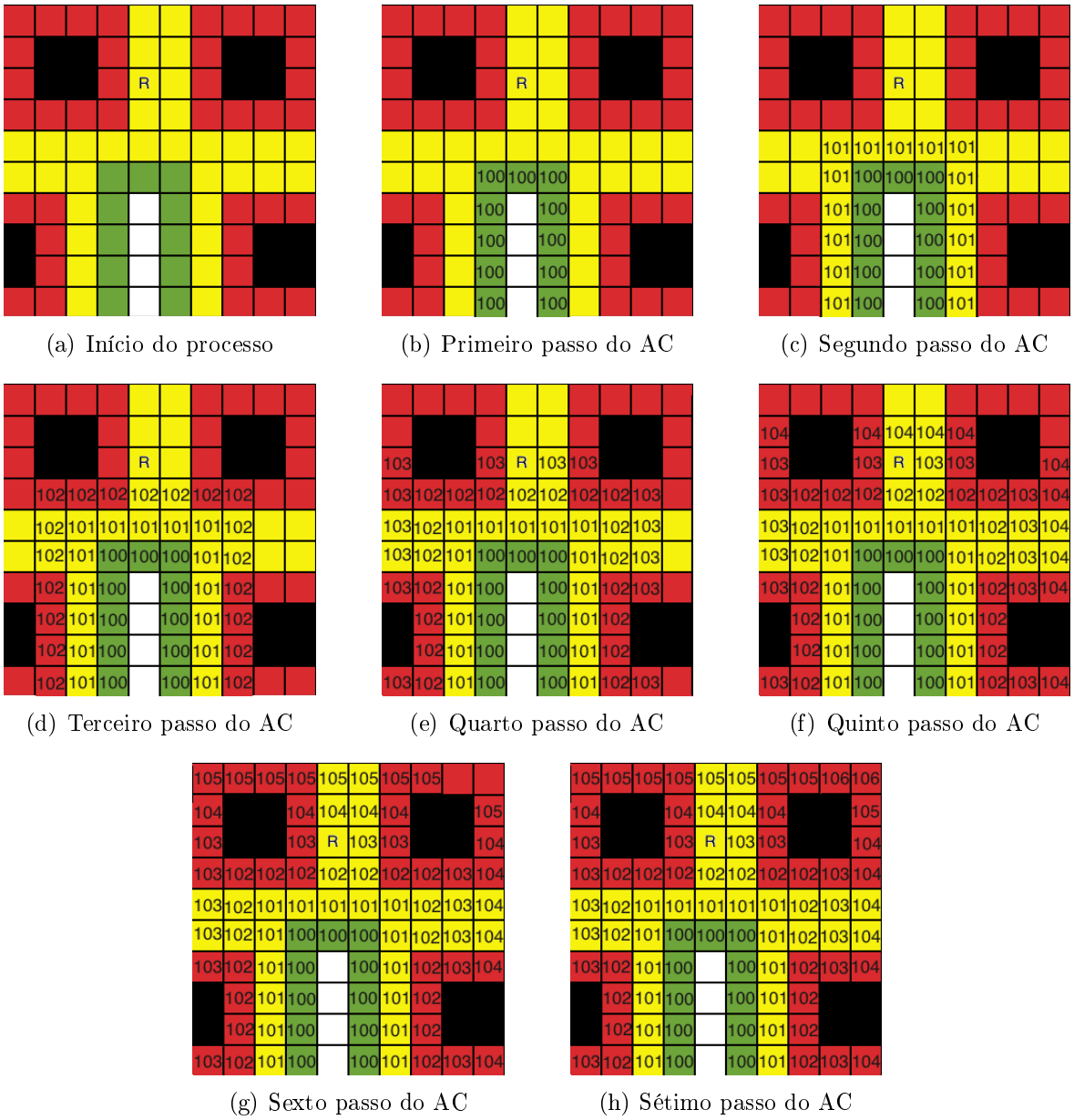
Primeiro AC da Fase 3 (Cálculo de *s2*)

Esse cálculo é realizado por uma regra de AC que utiliza a vizinhança de Moore e verifica se cada célula do reticulado se enquadra em pelo menos uma das situações identificadas a seguir como A e B:

- **Situacao A:** a regra do AC verifica se a regra atende 3 requisitos simultaneamente: (i) *s1*= Obstáculo-alargado (identificando que se trata de uma célula que pertence a área alargada após a primeira fase do algoritmo); (ii) *s2*=indefinido (ou seja, ainda não foi calculado o valor da distância para essa célula) e (iii) existe em sua vizinhança uma célula com *s1*=Livre (ou seja, existe pelo menos um vizinho da célula em questão que é uma célula livre). Caso a Situação A seja identificada, é definido um valor discreto V para essa célula ($s2 = V$). V é um valor arbitrário escolhido como parâmetro do algoritmo de forma que seja sempre maior que os valores de *s2* das células livres (por exemplo, em nossos experimentos escolhemos V=100 pois é impossível uma célula livre ter uma distância até a meta maior que 100, pelo tamanho dos reticulados utilizados).
- **Situação B:** a regra do AC verifica se a célula atende 4 requisitos simultaneamente: (i) *s1*= Obstáculo-alargado (ii) *s2*=indefinido (iii) não existe vizinho com *s1*=Livre e (iv) existe pelo menos um vizinho com o valor de *s2* definido. Caso a Situação B seja identificada, o valor de *s2* dessa célula se torna o menor valor de *s2* entre seus vizinhos mais um.

Para todos os outros casos que não se identificam com a Situação A ou B, a célula mantém o valor de *s2*. Dessa forma, as células livres continuam com o valor de distância calculados na fase 2 do algoritmo e as células que representam obstáculos reais continuam com o valor de *s2* indefinido. Essa regra é aplicada até que nenhuma célula do reticulado altere o valor de *s2*. Ao final de algumas iterações, todas as células de obstáculos alargados, que possuam pelo menos um caminho livre de obstáculos reais até uma célula livre, terão o valor de *s2* calculado. A Figura 4.7 ilustra a aplicação dessa regra que atualiza o valor de *s2* para todas as células com *s1*= Obstáculo-alargado, partindo-se do exemplo da Figura 4.6.b,como instante inicial de aplicação do novo AC, para o qual não é possível encontrar uma rota quando utilizado o modelo original descrito em [Behring et al. 2000]. Na Figura 4.7.a temos as células da área alargada (verdes, amarelas e vermelhas) com o valor de *s2* indefinido, pois apenas as células livres possuem esse valor definido ao final da fase 2 do algoritmo (células brancas, mas o valor de *s2* das mesmas foi omitido na figura para destacar o cálculo de *s2* nas áreas alargadas). Na Figura 4.7.b observamos o valor de *s2* após a aplicação da primeira iteração da nova regra. Vemos que, como consequência, as

células que estão nas bordas das áreas alargadas (ou seja, possuem uma vizinha que é célula livre) iniciam o cálculo assumindo o valor V (nesse exemplo, foi adotado V=100). Na figura 4.7.c observamos o valor de *s2* após a aplicação da segunda iteração da nova regra. Vemos que as células internas às áreas alargadas, que tenham pelo menos uma vizinha no passo anterior que teve o valor de *s2* calculado, passam a ter o valor de *s2* definido (nesse caso V+1). As figuras posteriores 4.7.d, 4.7.e, 4.7.f, 4.7.g e 4.7.h ilustram a difusão do cálculo de *s2* até que todas as células com *s1* = Obstáculo-alargado tenham o valor dessa métrica calculado.



(a) Início do processo (b) Primeiro passo do AC (c) Segundo passo do AC

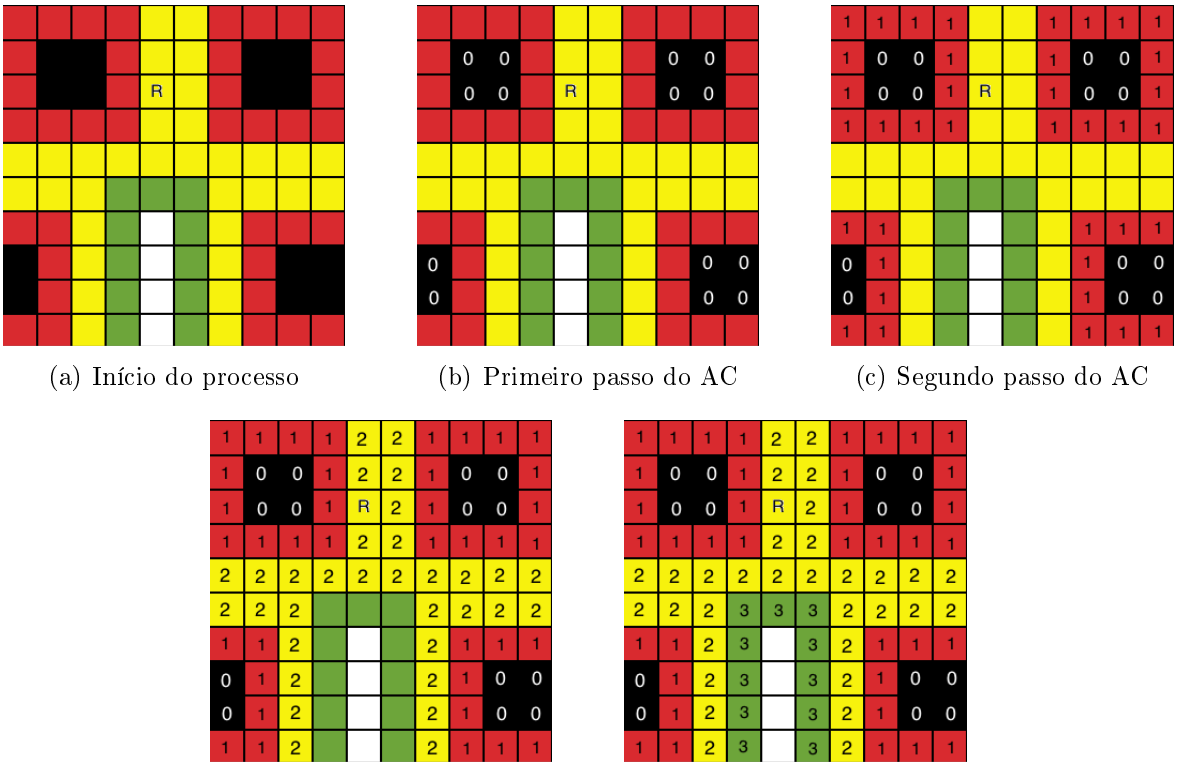
(d) Terceiro passo do AC (e) Quarto passo do AC (f) Quinto passo do AC

(g) Sexto passo do AC (h) Sétimo passo do AC

Figura 4.7: Execução do primeiro AC da fase 3.

Segundo AC da Fase 3 (Cálculo de *s3*)

O segundo AC, também utiliza a vizinhança de Moore e calcula a proximidade de um obstáculo real, armazenando esse valor no sub-estado *s3* das células. A regra de transição utilizada é: (i) se a célula for um obstáculo real (*s1* = Obstáculo), seu valor de proximidade é igual a zero (fazer *s3*=0), (ii) se a célula for um obstáculo alargado (*s1* = Obstáculo Alargado) e seu valor de *s3* ainda não foi calculado (*s3* = indeterminado), seu valor de proximidade é igual ao valor de maior proximidade entre os vizinhos mais um (fazer *s3* = max(*s3*) + 1), (iii) se a célula for livre (*s1* =Livre ou *s1*= Posição Inicial ou *s1*=Objetivo), o valor de proximidade permanece indefinido (manter *s3*=indefinido). A Figura 4.8 apresenta o valor de *s3* calculado para todas as células que estão em áreas alargadas, após a aplicação do segundo AC no mesmo cenário da Figura 4.7. No primeiro passo de tempo, mostrado na Figura 4.8.b, as células que são obstáculos reais são calculadas e seus valores de *s3* são inicializados como 0. A partir do segundo passo de tempo, os obstáculos alargados terão seus valores de *s3* calculados. A Figura 4.8.c mostra os obstáculos alargados mais próximos dos obstáculos reais tendo os seus valores de *s3* iguais a 1. As Figuras 4.8.d e 4.8.e mostram o restante do processo, sendo que esta última mostra os valores finais obtidos por *s3* após a aplicação do AC.



(a) Início do processo (b) Primeiro passo do AC (c) Segundo passo do AC

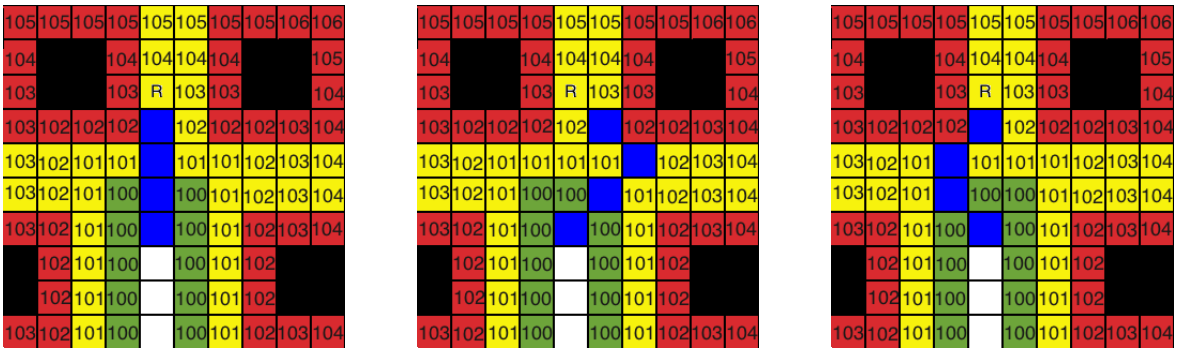
(d) Terceiro passo do AC (e) Quarto passo do AC

Figura 4.8: Execução do segundo AC da fase 3.

Após a aplicação das regras dos dois ACs que calculam as distâncias armazenadas em *s2* e *s3*, o algoritmo passa para a última fase, que é a obtenção da rota entre a posição

inicial e a meta. Nessa quarta fase, que define a rota final, o algoritmo diferencia se a terceira fase foi aplicada, significando que a posição inicial do robô está em uma área alargada. Caso a terceira fase não tenha sido aplicada, a obtenção da rota é similar ao modelo original em [Behring et al. 2000]. Caso a terceira fase tenha sido aplicada, a parte inicial da rota deve traçar um caminho para que o robô saia o mais rápido o possível da área alargada dos obstáculos. Para isso, o algoritmo inicia a partir da célula com a posição inicial do robô e escolhe a célula vizinha que contém o menor valor de *s2* (ou seja, a menor distância até uma célula livre fora dos obstáculos alargados). Caso haja empate entre os vizinhos em relação a *s2*, o algoritmo escolhe a célula com maior valor de *s3*. Ou seja, dentre as células empatadas em *s2*, escolhe aquela que está mais distante de um obstáculo real. A definição da rota é feita dessa forma até que o algoritmo selecione uma célula com *s1* =Livre. A partir desse ponto, a rota já está fora das áreas alargadas e o algoritmo volta a calcular a rota usando o mesmo princípio do modelo original: escolher sempre a célula vizinha com menor valor de distância até a meta, dado pelo valor do sub-estado *s2* de todas as células com *s1* =Livre. Nesse caso, caso haja empate, escolhe uma das vizinhas empatadas aleatoriamente. Na Figura 4.7.h, temos um exemplo de ambiente com os valores de *s2* e *s3* calculados pelos ACs da fase 3. Os valores de *s2*, que representam a distância de cada célula até uma célula livre, estão apresentados diretamente nas células e os valores de *s3*, que representam a distância de cada célula ate um obstáculo real, estão representados pelas cores das células, sendo que podemos considerar: células pretas (*s3*=0), células vermelhas (*s3*=1), células amarelas (*s3*=2) e células verdes (*s3*=3). No início do cálculo da rota para esse cenário ilustrado na Figura 4.7.h, as vizinhas da célula onde o robô esta posicionado são analisadas para se definir o primeiro passo do robô. Nesse exemplo, existem três possíveis células para esse primeiro deslocamento, pois todas tem *s2*=102. Dessas células, duas são amarelas (*s3*=2) e uma é vermelha (*s3*=1). Nesse caso, o algoritmo selecionará uma das células amarelas, para que o robô passe o mais distante possível de um obstáculo real. Suponha que o algoritmo selecione a célula mais a esquerda. No próximo passo, o robô analisará as vizinhas dessa célula com *s2*=102 para novamente escolher a vizinha com menor valor de *s2* como próximo passo, sendo que novos desempates serão realizados utilizando-se o valor de *s3*. A Figura 4.9 ilustra diversos possíveis caminhos obtidos a partir do cenário da Figura 4.7.h, até que a rota alcance uma célula livre (cor branca), caso a decisão de desempate entre células (iguais em *s2* e em *s3*) se dê de forma aleatória. Entretanto, no modelo também é possível definir escolhas determinísticas nessa situação. Por exemplo, se a escolha for feita pela célula que provoca um menor número de rotações, o caminho da Figura 4.9.a seria selecionado pelo algoritmo. Por outro lado, se a escolha fosse determinística pela primeira célula no sentido horário, a rota seria a da Figura 4.9.b. Se o sentido fosse o anti-horário, seria a rota da Figura 4.9.c. A escolha do tipo de desempate a ser utilizado é um parâmetro de entrada do algoritmo. A partir do ponto em que a rota que está sendo construída

alcança uma célula livre fora da área de alargamento dos obstáculos, o algoritmo passa a calcular o restante da rota conforme o modelo original em [Behring et al. 2000], ou seja considerando apenas o valor de *s2* para as células livres.



(a) (b) (c)

Figura 4.9: Caminhos possíveis para encontrar a saída dos obstáculos pelo modelo modificado.

A Figura 4.10 apresenta o cálculo de uma rota completa calculada pelo algoritmo no novo modelo. Nessa rota, a parte roxa é calculada utilizando-se os valores de *s2* e *s3* das células nos obstáculos alargados. A parte azul representa a parte da rota calculada utilizando-se unicamente os valores de s2 das células livres.

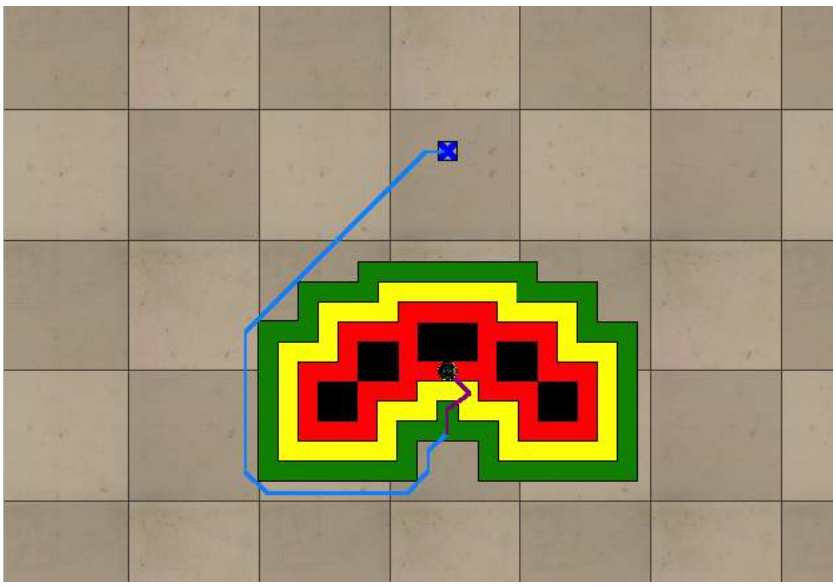


Figura 4.10: Rota encontrada partindo de dentro do obstáculo.

O algoritmo modificado com as duas modificações pode ser visto na Figura 4.11.

```
t=0
//Quantidade de passos de crescimento
passos = 4
imagem = recebe_imagem()
reticulado = discretiza_imagem(imagem)
reticulado_final = aumenta_obstaculos(reticulado, passos)
while reticulado_final[meta.x][meta.y] == Robo do
      //Resolução do primeiro problema
      t = t+1
      if t == num_iter then
            dentro_obstaculo = false
            imagem = recebe_imagem()
            reticulado = discretiza_imagem(imagem)
            reticulado_final = aumenta_obstaculos(reticulado, passos)
            while reticulado_final[inicial.x][inicial.y] == Livre do
                  novo_reticulado = calcula_distancia(reticulado_final)
                  //Não encontrou caminho
                  if novo_reticulado == reticulado_final then
                        dentro_obstaculo = true
                        break;
                  else
                        reticulado_final = novo_reticulado
                  end-if
            end-while
            //Resolução do segundo problema
            if dentro_obstaculo == true then
                  //Primeiro gradiente
                  while reticulado_final[inicial.x][inicial.y] == Livre do
                        novo_reticulado = distancia_celula_livre(reticulado_final)
                        reticulado_final = novo_reticulado
                  end-while
                  //Segundo gradiente
                  for i=0; i<passos; ++i
                        proximidade = calcula_prox(reticulado_final, proximidade)
                  end-for
            end-if
            t = 0
      end-if
      reticulado_final = movimenta_robo(robo, reticulado_final)
end-while
```

Figura 4.11: Pseudo-código com as soluções propostas.

4.4 Discussão

Ao final das investigações do trabalho [Behring et al. 2000], foi proposto um novo modelo baseado na Difusão da Distância à Meta que corrige os problemas encontrados no trabalho original da abordagem. O primeiro problema que consistiu na posição final do robô ser distante da posição que ele deveria chegar ao final da movimentação - devido às leis da cinemática e dinâmica - foi melhorado através do re-cálculo do planejamento de caminhos após uma quantidade fixa de passos, assim o robô sabe em que posição realmente está naquele momento e não a posição que ele estima estar. O segundo problema encontrado durante os experimentos em simulação foi que quando o robô se encontrava dentro de uma "armadilha de obstáculos", o planejamento não era possível de ser realizado

pelo modelo original. Cabe destacar que este problema ocorre com mais frequência após a primeira modificação, pois quando o robô recebe uma nova imagem do ambiente, existe a possibilidade dele se encontrar naquele momento dentro de um obstáculo que cresceu. Para a correção deste problema, foi proposta uma adaptação onde ocorre também a difusão da distância dentro dos obstáculos, assim se tornou possível que o robô saia de dentro de um obstáculo que cresceu.

Com estas modificações, o objetivo inicial proposto para a investigação deste modelo foi atingido, pois foi feita a investigação de um modelo simples de ser implementado, onde foi possível aprender os conceitos necessários para o planejamento de caminhos e movimentação de robôs autônomos, além de ser possível de testar em um robô real da arquitetura e-puck, que existia no laboratório de computação Bio-inspirada da Universidade Federal de Uberlândia. Porém, apesar de possível de ser implementado em robôs da arquitetura e-puck, estes experimentos não foram realizados devido à falta de um ambiente adequado para a aquisição de imagens da mesa de experimentos a cada passo de tempo. Assim, como trabalho futuro para este modelo, pode-se montar um ambiente adequado para estes experimentos, além da colaboração com o grupo de processamento digital de imagens para agregar conhecimento para a fase de pré-processamento do algoritmo, ou seja, a parte de aquisição da imagem do ambiente.

Decidimos que o próximo modelo que seria investigado não deveria depender de um sistema de aquisição e processamento de imagens, para que fosse possível sua implementação em robôs e-puck reais. Uma abordagem com essa característica é a Abordagem por Regra de Atualização Local, que, por não ter conhecimento prévio do ambiente, faz uma escolha local de acordo com os dados dos sensores naquele passo de tempo. A investigação detalhada desta abordagem será mostrada no Capítulo 5.

Capítulo 5

Investigação sobre a Abordagem por Regra de Atualização Local

Buscando um segundo modelo para investigação com características mais próximas daquelas inerentes aos ACs, neste capítulo será detalhado um modelo que faz parte da abordagem por Regra de Atualização Local. A escolha deste modelo foi feita devido a três características encontradas: a primeira é que o tipo de regras dos autômatos celulares desta abordagem foi a mais próxima de um comportamento importante nos ACs, que é uma modificação local causar uma dinâmica global no reticulado; a segunda característica interessante é a característica cooperativa incorporada ao modelo quando usado mais de um robô; por fim, os autores fizeram testes com robôs reais da arquitetura e-puck, que estávamos adquirindo para nosso laboratório, assim seria possível reproduzir os experimentos feitos pelos autores.

Este capítulo começa detalhando o modelo original de Ioannidis e colaboradores [Ioannidis et al. 2008] que é a base de nossa implementação. Depois serão apresentadas as modificações necessárias no modelo original tendo em vista a implementação em um ambiente contendo somente um robô, além dos resultados obtidos. Posteriormente, serão descritas as alterações das regras para um único robô tendo como objetivo a correção de problemas encontrados em simulação. A quarta seção analisa os resultados encontrados no modelo com um time de robôs agindo de forma cooperativa. Por fim, será mostrada a conclusão dos experimentos realizados com este modelo.

5.1 Modelo de Ioannidis e colaboradores (2008)

A principal característica da abordagem que classificamos como "Regra de Atualização Local" consiste na tomada de decisão local sobre a movimentação a ser realizada a cada passo discreto de tempo. Para isso, os modelos utilizam a leitura dos sensores a cada passo de tempo para identificar a vizinhança da célula onde está o robô, e aplicam uma

regra de atualização local que decide qual será o próximo movimento do robô.

Foram encontrados trabalhos de dois grupos de pesquisa distintos que utilizam esta abordagem para o planejamento de caminhos. O grupo de Akbarimajd e colaboradores, que foram os primeiros a propor esta abordagem, na qual focam em encontrar um caminho qualquer entre um ponto inicial e uma meta, para um ou mais robôs, utilizando os sensores para construir a vizinhança do robô. O grupo de Ioannidis e colaboradores tem como objetivo também o controle de formação, ou seja, os robôs não devem sair de uma formação específica enquanto caminham pelo ambiente. Devido à natureza cooperativa do trabalho do segundo grupo, este foi escolhido para a nossa pesquisa.

Em nossa investigação, nosso foco foi no trabalho descrito em [Ioannidis et al. 2008], no qual o ambiente de movimentação do robô é um espaço euclidiano bidimensional, que contém células quadradas e idênticas, com cada lado de tamanho t. Os autores em [Ioannidis et al. 2008] verificaram que o tamanho das células é fortemente dependente da qualidade dos sensores dos robôs. Com isso, o tamanho t de cada célula não deve ser pequeno a ponto de tornar o processo de movimentação muito lento e com alta exigência de memória, e nem tão grande a ponto do robô não reconhecer um obstáculo próximo. Em [Ioannidis et al. 2011b], os autores mostram os experimentos utilizados para definir um tamanho apropriado para as células. Foram utilizados os robôs e-puck [École Polytechnique Fédérale de Lausanne EPFL 2013], e neles, os sensores de infra-vermelho retornam valores inteiros que quanto mais próximos de 0, mais distante o robô está de um obstáculo. Em [Ioannidis et al. 2011a], os autores utilizam o tamanho de célula igual a 2 centímetros. Uma vez definido o tamanho da célula, é possível que o robô seja maior do que as dimensões da célula. Nesse caso, o robô ocuparia realmente mais de uma célula, mas no ponto de vista do algoritmo, cada robô ocupa uma única célula, que é aquela em que o seu centro está localizado.

A cada passo de tempo, o robô poderá realizar dois tipos de movimentação: (i) deslocar-se para uma célula vizinha, mantendo sua orientação atual, ou (ii) realizar uma rotação sobre seu eixo, mantendo sua posição atual. Quando a ação é de deslocamento, a distância percorrida pelo robô será a distância entre a célula em que ele está e a célula vizinha, ou seja, um passo de deslocamento equivale a percorrer a largura de uma célula (t), se o deslocamento é para as células norte, sul, leste ou oeste, e equivale a $\sqrt{2t}$, se o deslocamento for para as células das vizinhanças diagonais. Por outro lado, quando a ação é uma rotação, o robô efetuará um giro com 4 valores possíveis de ângulo: $90°$, $45°$, $-45°$ ou $-90°$.

O algoritmo de planejamento de caminhos proposto por Ioannidis e colegas é dividido em três fases: (i) construção da vizinhança das células, (ii) decisão da regra do autômato celular a ser aplicada e (iii) aplicação da regra tendo como consequência a alteração do estado da célula central e a efetivação da ação de movimentação do robô.

No modelo de autômato celular proposto em [Ioannidis et al. 2008], cada célula pode

assumir um dentre os $R + 2$ estados possíveis, sendo R o número de robôs no ambiente. O estado 0 representa uma célula vazia, os estados de 1 a R representam que a célula está ocupada por um robô identificado pelo mesmo número do estado e o estado $R + 1$ representa uma célula com obstáculo. O tipo de vizinhança utilizada é a vizinhança de Moore de raio 1.

A vizinhança de cada robô será construída a cada passo de tempo através da leitura de seus sensores. Assim, como arquiteturas *e-puck* foram utilizadas em [Ioannidis et al. 2008], varre-se os oito sensores do robô e de acordo com sua posição, decide-se se existe um obstáculo em determinado vizinho. Como o resultado da leitura é um valor inteiro, foi determinado um *threshold* para decidir se uma vizinha contém ou não um obstáculo. Para o algoritmo, a vizinhança é um vetor de nove posições, que conterá o valor 0 caso a célula seja livre e R+1 caso a célula contenha um obstáculo. As posições no vetor para cada célula vizinha, podem ser vistas na Figura 5.1.

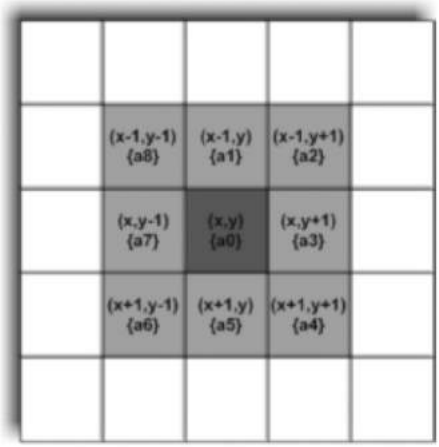


Figura 5.1: Posições de cada célula da vizinhança no vetor de vizinhos. Sendo *a0* a primeira posição e *a8* a nona e última posição. [Ioannidis et al. 2011b]

Tendo a vizinhança do robô, para a aplicação da regra de movimentação, é necessário também saber o ângulo de rotação do robô θ_R. Foi definido que o robô poderá estar em cinco orientações distintas, com a diferença entre as posições igual a 45°. O ângulo 0° é definido como a direção original que o robô deve se movimentar. Por exemplo, se no início da aplicação do algoritmo o robô deve se deslocar para o norte (direção sul-norte), teremos a seguinte representação para o ângulo θ_R: 0° (direcionado para a célula a_1), 45° (direcionado para a célula a_2), 90° (direcionado para a célula a_3), $-45°$ (direcionado para a célula a_8), $-90°$ (direcionado para a célula a_7).

A regra de transição do autômato celular define a movimentação do robô pelo ambiente, estabelecendo a próxima ação a cada passo de tempo, além de atualizar o valor das células. A regra de transição depende da configuração da vizinhança do robô e é dividida em dois tipos: as regras para evitar colisões com obstáculos e as regras de controle de formação. Basicamente, o algoritmo define prioritariamente, qual tipo de regra deve ser

aplicado a cada instante. Após a leitura do ambiente e construção da vizinhança, caso haja algum obstáculo em alguma célula vizinha, o algoritmo aplicará uma regra de desvio de obstáculos; caso o robô não encontre nenhum obstáculo em volta, utiliza a regra de controle de formação. Além disso, no modelo discutido em [Ioannidis et al. 2008], como o problema abordado é cooperativo e existe mais de um robô no ambiente, adota-se o conceito de robô mestre e robô escravo:

(i) Cada robô identifica inicialmente sua vizinhança e qual tipo de regra será aplicada (desvio de obstáculo ou controle de formação).

(ii) Se for identificado um obstáculo na vizinhança, o robô simplesmente aplicará a regra adequada de desvio, sem se preocupar com a movimentação do restante do time de robôs.

(iii) Por outro lado, se nenhum obstáculo for identificado, um robô escravo deverá enviar a sua posição e orientação para o robô mestre, que fará a seleção da regra de controle de formação adequada e enviará de volta ao escravo qual a ação a ser executada (resultado da aplicação da regra). Caso o robô seja o mestre, ele irá receber a posição e orientação dos robôs escravos, além de obter sua própria vizinhança a partir da leitura de seus sensores. A partir dessas informações, o robô mestre irá decidir qual a regra a ser aplicada e o próximo movimento de cada robô do time, incluindo a si mesmo. Só depois de receberem as ações de controle de formação a serem efetuadas, os robôs escravos irão efetuar os respectivos movimentos.

5.1.1 Regras de Desvio de Obstáculo

O primeiro conjunto de regras se refere a cenários em que algum obstáculo é identificado na vizinhança pelos sensores do robô. Nessa situação, o desvio desses obstáculos na decisão de próximo movimento é prioritário em relação à posição do robô na manutenção da formação do time. Por isso, o robô decide autonomamente a regra a ser disparada de acordo com o cenário. Ou seja, mesmo que seja um robô escravo, a decisão será local sem aguardar o comando do mestre.

Podemos dividir as regras de desvio de obstáculos em 2 tipos: (i) regras que deslocam o robô em uma célula; (ii) regras que rotacionam o robô.

As regras que provocam um deslocamento no robô são na verdade compostas por pares de regras: uma regra é executada para a célula central da vizinhança que contém o robô ao centro (e que deixará de ter o robô no próximo instante) e a segunda regra é executada para a célula central da vizinhança que está livre ao centro e que, devido à orientação do robô, receberá o mesmo no próximo passo de tempo. A Figura 5.2 exemplifica pares de cenários de deslocamento. Por exemplo, o par de cenários A e B exemplifica a situação em que o robô deve se deslocar para o norte, a sua orientação atual é 0° e não existe nenhum obstáculo na vizinhança que impeça o robô de continuar a se deslocar na direção desejada.

O cenário A exemplifica o que ocorrerá com a vizinhança cuja célula central contém o robô: no próximo passo de tempo, ela deverá se tornar livre. O cenário B exemplifica o que irá ocorrer com a vizinhança cuja célula central está livre e está na direção atual de deslocamento do robô: no próximo passo de tempo o estado deverá se tornar *Robô*. Cabe destacar que no modelo de [Ioannidis et al. 2008], também é necessário verificar se as células a_2 e a_8 estão livres para que o robô não fique preso entre dois obstáculos. Os cenários C e D exemplificam uma situação em que o robô está rotacionado em $-90°$ e com um obstáculo na célula a_1, que impede que ele volte sua orientação para $0°$. Nesse caso, o robô deve permanecer na mesma orientação ($-90°$) e se deslocar para a célula à direita (a_3). De forma similar, os cenários E e F exemplificam uma situação em que o robô está rotacionado em $90°$ e com obstáculos nas células a_1 e a_2. Assim, o robô permanece na mesma orientação ($90°$) e se desloca para a célula à esquerda (a_7). Além da alteração do estado da célula central, o disparo dessas regras também ocasiona a dinâmica de deslocamento do robô, para que de fato o mesmo percorra uma distância equivalente ao tamanho de 1 célula. Embora os cenários apresentados na Figura 5.2 auxiliem o entendimento de como as regras devem ser aplicadas aos pares em cenários de deslocamento, eles de fato não representam fielmente as regras a serem aplicadas, pois em uma regra de transição de um AC a saída se refere unicamente ao novo valor da célula central. Assim, as regras correspondentes aos cenários A a F da Figura 5.2 são representadas linearmente na Figura 5.3.

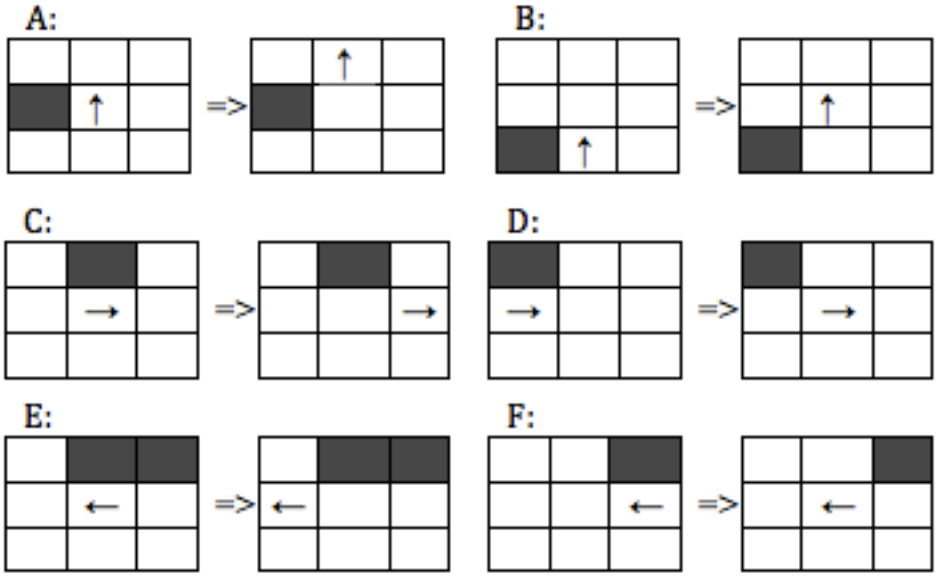


Figura 5.2: Subconjunto de pares de regras para deslocamento durante a fase de desvio de obstáculos.

As regras que forçam uma rotação, são aplicadas em cenários nos quais o robô se depara com dois tipos de situação: (a) está se deslocando na orientação original ($0°$) e se depara com um obstáculo que impede que ele mantenha a direção de deslocamento; (b)

Células no tempo t										Estado de a0 no tempo t+1	
a0	a1	a2	a3	a4	a5	a6	a7	a8	Θ_t	a0	Θ_{t+1}
r	f	f	f	f	f	f	o	f	0°	f	0°
f	f	f	f	f	r	o	f	f	0°	r	0°
r	o	f	f	f	f	f	f	f	-90°	f	-90°
f	f	f	f	f	f	f	r	o	-90°	r	-90°
r	o	o	f	f	f	f	f	f	90°	f	90°
f	f	o	r	f	f	f	f	f	90°	r	90°

Figura 5.3: Subconjunto de regras de deslocamento para o desvio de obstáculos, onde r é uma célula que contém um robô, f é uma célula livre e o é uma célula que contém um obstáculo.

está se deslocando em uma orientação que indica uma rotação anterior (diferente de 0°) e se depara com uma vizinhança que indica que o obstáculo que impedia o deslocamento na direção original já foi contornado.

Como exemplo do primeiro tipo de situação, suponha que o robô está se deslocando em linha reta para o norte e os sensores indicam um obstáculo a frente (célula a_1). Nesse caso, o robô deverá fazer uma rotação, de 90° ou −90°, para conseguir desviar deste obstáculo. A Figura 5.4 apresenta alguns cenários nos quais o robô realiza uma rotação após a aplicação da regra de transição, sendo que nos três primeiros o robô rotaciona −90° e nos três últimos 90°. Em todos os cenários iniciais da Figura 5.4, o robô se encontra à 0° e identifica um obstáculo na célula a_1. A idéia geral desses cenários é que ao se deparar com um obstáculo na célula a_1: (i) se as células a_2 e a_3 estiverem livres, o robô irá fazer uma rotação de −90° para que no próximo passo ele se desloque para a célula a_3; (ii) se a célula a_2 ou a célula a_3 estiver com um obstáculo, e as células a_7 e a_8 estiverem livres, o robô irá fazer uma rotação de 90° para que no próximo passo ele se desloque para a célula a_7; (iii) se as células a_2 e a_7 tiverem obstáculos e a_3 estiver livre, o robô irá fazer uma rotação de −90° para que no próximo passo ele se desloque para a célula a_3; (iv) se as células a_3 e a_8 tiverem obstáculos e a_7 estiver livre, o robô irá fazer uma rotação de 90° para que no próximo passo ele se desloque para a célula a_7. Os cenários ilustrados na Figura 5.4 apenas representam um subconjunto dentre todos os possíveis que se enquadram nas 4 situações acima. A Figura 5.5 apresenta as regras de transição associadas a esses 6 cenários.

O segundo tipo de situação que provoca uma rotação do robô, ocorre quando ele está em uma orientação diferente de 0°, sinalizando que em algum momento anterior foi necessário rotacionar, e não mais identifica um obstáculo que impeça o retorno à orientação original. Assim, suponha que o robô estava se deslocando originalmente em linha reta para o norte, mas no momento está com angulo de 90° e os sensores indicam que a célula

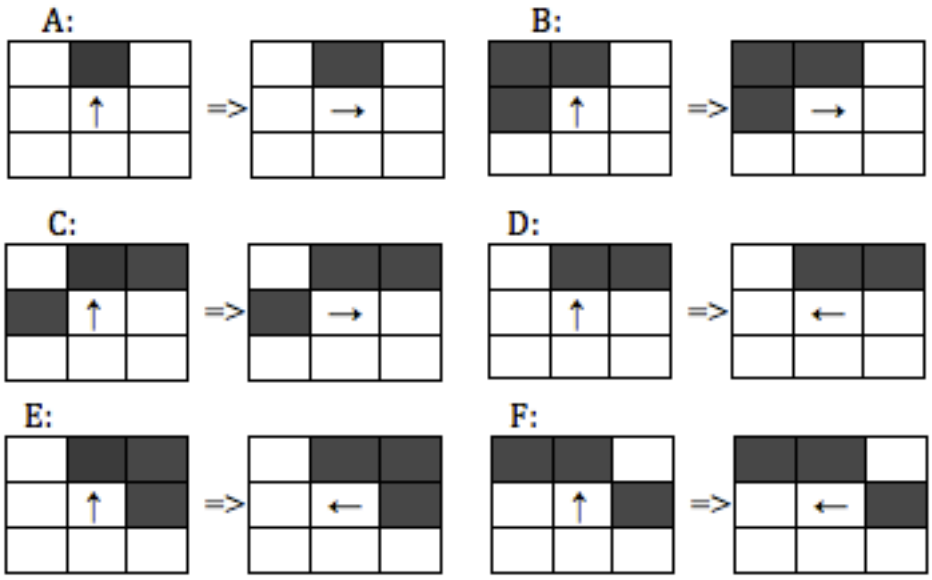


Figura 5.4: Subconjunto de regras onde ocorre uma rotação durante a fase de desvio de obstáculos.

Células no tempo t										Estado de a0 no tempo t+1	
a0	a1	a2	a3	a4	a5	a6	a7	a8	Θ_t	a0	Θ_{t+1}
r	o	f	f	f	f	f	f	f	0°	r	-90°
r	o	f	f	f	f	f	o	o	0°	r	-90°
r	o	o	f	f	f	f	o	f	0°	r	-90°
r	o	o	f	f	f	f	f	f	0°	r	90°
r	o	o	o	f	f	f	f	f	0°	r	90°
r	o	f	o	f	f	f	f	o	0°	r	90°

Figura 5.5: Subconjunto de regras de rotação para o desvio de obstáculos.

a_1 está livre. A Figura 5.6 apresenta 3 cenários nos quais o robô rotaciona de forma a voltar à orientação inicial para o norte (0°). Cenários similares podem ocorrer quando o robô está rotacionado a $-90°$ e também não mais identifica o obstáculo que impedia sua progressão ao norte. A Figura 5.7 apresenta as regras linearizadas correspondentes aos 3 cenários da Figura 5.6.

Em [Ioannidis et al. 2008], os autores ilustram as regras de transição de forma linearizada, conforme as apresentadas nas figuras 5.3, 5.5 e 5.7. Entretanto, se as regras forem escritas dessa forma individual, seria necessário desenvolver uma listagem exaustiva de todas as configurações possíveis de ocorrer na vizinhança de uma célula. Considerando-se apenas as regras que têm o robô na célula central, o robô pode estar em 5 orientações distintas e as 8 células vizinhas podem assumir 2 valores diferentes (Livre, Obstaculo).

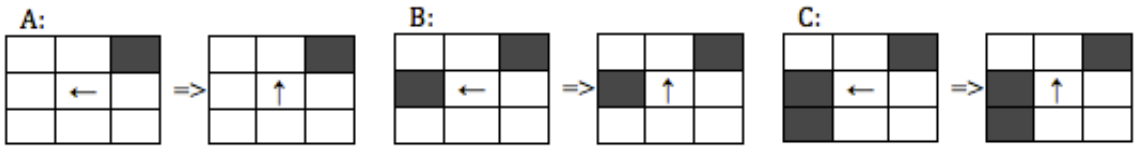


Figura 5.6: Subconjunto de regras onde ocorre a rotação para o ângulo 0° durante a fase de desvio de obstáculos.

Células no tempo t										Estado de a0 no tempo t+1	
a0	a1	a2	a3	a4	a5	a6	a7	a8	Θ_t	a0	Θ_{t+1}
r	f	o	f	f	f	f	f	f	90°	r	0°
r	f	o	f	f	f	f	o	f	90°	r	0°
r	f	o	f	f	f	o	o	f	90°	r	0°

Figura 5.7: Subconjunto de regras de retorno ao ângulo 0° durante o desvio de obstáculos.

Nesse caso, seriam $5 * 2^8$ (1280) cenários diferentes a serem retratados nas regras. Assim, adotamos em nossa implementação uma representação de regra totalística, que generaliza diversos cenários em uma única descrição. Além de facilitar a visualização dos movimentos nos desvios de obstáculos, essa forma de implementação viabilizou o embarque do código nos robôs reais, onde há uma severa restrição de memória. A Figura 5.8 apresenta a regra de transição implementada para o modelo original de Ioannidis e colegas.

5.1.2 Regras de Controle de Formação

O problema a ser resolvido pelo modelo em [Ioannidis et al. 2008], é o deslocamento em uma mesma direção de um time de robôs que deve manter uma formação pré-estabelecida. Nos trabalhos de Ioannidis e colegas, duas formações são apresentadas, a formação em linha reta e a formação triangular. Na formação em linha, os robôs devem manter, sempre que possível, uma distância fixa em células horizontais entre eles. Para a formação triangular, também é necessário manter uma distância vertical entre os robôs. Para simplificarmos a explicação, vamos assumir somente a formação em linha reta, mas para a formação triangular é semelhante.

Independente da formação escolhida, temos a existência de robôs mestres e robôs escravos. Cada robô mestre comanda no máximo dois robôs escravos, seus vizinhos à esquerda e à direita, se existirem. Cada conjunto de no máximo três robôs é chamado de grupo e devem manter sua formação de forma independente. Durante este trabalho, foram investigadas formações com três robôs (um mestre e dois escravos), mas nos trabalhos de Ioannidis e colegas foram apresentados exemplos com até cinco robôs, ou seja, dois grupos com um mestre para cada grupo. O robô mestre é o responsável por manter a formação do grupo, verificando a sua própria posição e de seus robôs escravos podendo decidir o

Célula Central	Ângulo	Vizinhança	Nova Célula Central	Novo Ângulo
LIVRE	0°	a5 = ROBO & a3 & a7 = LIVRE	ROBO	0°
		Caso Contrário	LIVRE	0°
	90°	a3=ROBO & a5 = LIVRE & a2 = OBSTACULO	ROBO	-90°
		Caso Contrário	LIVRE	-90°
	-90°	a7=ROBO & a5 = LIVRE & a8 = OBSTACULO	ROBO	90°
		Caso Contrário	LIVRE	90°
OBSTACULO	Qualquer	Qualquer Vizinhança	OBSTACULO	Qualquer
ROBO	0°	a1 & a2 & a8 = LIVRE	LIVRE	0°
		a1 = OBSTACULO & a2 & a3 = LIVRE	ROBO	-90°
		a1 & (a2 OR a3) = OBSTACULO & a7 & a8 = LIVRE	ROBO	90°
		a1 & a7 = OBSTACULO & a3 = LIVRE	ROBO	-90°
		a1 & a3 = OBSTACULO & a7= LIVRE	ROBO	90°
		a8 = OBSTACULO & a3 = LIVRE	ROBO	-90°
		a2 = OBSTACULO & a7 = LIVRE	ROBO	90°
	90°	a1 = OBSTACULO & a7 & a6 = LIVRE	LIVRE	90°
		a1 = LIVRE	ROBO	0°
	-90°	a1 = OBSTACULO & a3 & a4 = LIVRE	LIVRE	-90°
		a1 = LIVRE	ROBO	0°

Figura 5.8: Regras de transição relativas ao modelo original de Ioannidis e colegas.

próximo passo de cada robô com o objetivo de manter a formação ou retornar a ela caso tenha sido perdida.

Cada robô tem autonomia para decidir qual regra de desvio de obstáculos irá aplicar caso exista algum impedimento na vizinhança, escolhendo a regra de acordo com a Figura 5.8. Porém, caso não exista obstáculo, os robôs de um mesmo grupo devem trocar informações para que seja aplicada a regra de controle de formação que fará com que os robôs mantenham a formação original ou retornem a ela o mais rápido o possível, caso a mesma tenha sido perdida.

Se, durante o processo de movimentação do time, algum dos robôs encontra algum obstáculo, ele irá aplicar uma regra de desvio de obstáculos, com isso, o grupo de robôs pode perder a formação desejada caso algum dos robôs saia de sua coluna original. Como os robôs devem se deslocar sempre em uma mesma direção, ou seja, em uma mesma coluna, é necessário que cada robô saiba sua posição horizontal original, que é denominada hor_r para um dado robô r. Em acréscimo à posição horizontal original, também é necessário que o robô saiba a distância que deve preservar dos robôs vizinhos para que a formação seja sempre mantida. Esta distância, denominada *dist* é definida como $dist = |x_{r1} - x_{r2}|$, onde x_{r1} e x_{r2} são as posições horizontais de dois robôs vizinhos no grupo.

As regras de controle de formação visam manter o valor de x_r ou seja, a coluna em

que o robô está neste passo de tempo, igual a hor_r que é a coluna original do robô. Para isso, caso o robô esteja à esquerda de sua coluna original, ele irá se movimentar para a sua diagonal superior direita, caso ele esteja à direita de sua coluna original, ele irá se movimentar para a sua diagonal superior esquerda. Finalmente, caso ele já esteja na sua coluna original, ele deverá ir para a célula à frente. Todas as regras de controle de formação podem ser vistas na Figura 5.9.

Cabe aqui destacar uma diferença na forma da descrição das regras de controle de formação, em relação às regras de desvio de obstáculos para deslocamento descritas na Seção 5.1.1. Conforme apresentado na figura, as regras de controle de formação também são apresentadas aos pares, sendo o primeira regra do par aquela que se refere à vizinhança da célula central que no momento o robô está posicionado e a segunda se refere à vizinhança cuja célula central irá receber o robô. Assim, as duas primeiras regras da Figura 5.9 se referem à situação em que o robô irá se deslocar para a frente, pois ele já está na coluna correta (hor_r - $\text{x}_r = 0$). A terceira e quarta regras da Figura 5.9 se referem à situação em que o robô está à esquerda da coluna correta ($\text{hor}_r > \text{x}_r$) e nesse caso o robô irá se deslocar para a diagonal superior direita (da célula a6 para a célula a0). Entretanto, diferentemente do que foi apresentado nas regras de desvio de obstáculo, a ação decorrente da aplicação dessas duas regras se refere a três movimentos: uma rotação (-45°), seguida de um deslocamento (a6 para a célula a0), seguida de uma nova rotação (45°). Dessa forma, o robô parte de uma orientação de 0°, gira 45° no sentido horário, se desloca 1 passo na diagonal, gira 45° no sentido anti-horário e volta para a orientação de 0°. Se adotássemos a mesma representação utilizada nas regras de desvio de obstáculos, esse movimento seria efetuado pela aplicação de 4 regras, aplicadas em 3 passos de tempo sucessivos: no primeiro passo a aplicação de uma regra de rotação; no segundo passo a aplicação do par de regras de deslocamento e no terceiro passo a aplicação de uma regra de rotação. Seguimos essas representações diferentes pelo fato dos autores fazerem da mesma forma no artigo original. As duas últimas regras da Figura 5.9 descrevem a situação em que o robô irá se deslocar na diagonal superior esquerda e são similares à situação descrita para o deslocamento na diagonal superior direita.

Outro conceito importante para o processo de cooperação entre os robôs é o de troca de posições. Caso dois robôs se aproximem após pelo menos um ter desviado de algum obstáculo, o próximo passo que for aplicar uma regra de controle de formação, os dois robôs que se aproximaram irão trocar entre si sua coluna original (hor_r). Assim, o robô que estava à direita, irá para a esquerda e o robô que estava à esquerda irá para a direita, aplicando as mesmas regras mostradas na Figura 5.9. É importante salientar que o antigo robô escravo, independente se estava à esquerda ou à direita, irá assumir a coluna do robô mestre e também sua posição como o mestre da formação, consequentemente o antigo robô mestre irá se tornar um robô escravo após a mudança de posições.

Por fim, caso seja necessário utilizar uma regra de controle de formação, devido a

Células no tempo t											Estado de a0 no tempo t+1	
Caso	a0	a1	a2	a3	a4	a5	a6	a7	a8	Θ_t	a0	Θ_{t+1}
$hor_r - x_r = 0$	r	f	f	f	f	f	f	f	f	0°	f	0°
$hor_r - x_r = 0$	f	f	f	f	f	r	f	f	f	0°	r	0°
$hor_r - x_r > 0$	r	f	f	f	f	f	f	f	f	0°	f	0°
$hor_r - x_r > 0$	f	f	f	f	f	f	r	f	f	0°	r	0°
$hor_r - x_r < 0$	r	f	f	f	f	f	f	f	f	0°	f	0°
$hor_r - x_r < 0$	f	f	f	f	r	f	f	f	f	0°	r	0°

Figura 5.9: Todas as regras possíveis para a controle de formação com o objetivo de retomar a formação original o mais rápido possível.

não existir nenhum obstáculo na vizinhança do robô, o processo de cooperação ocorre primeiramente trocando informações entre os robôs para que se decida qual o próximo passo que deverá ser aplicado. Se o robô que deverá aplicar a regra de controle de formação for um robô escravo, ele irá enviar uma mensagem para o robô mestre de seu grupo contendo seu identificador, sua posição horizontal e sua posição vertical. O primeiro campo é necessário para saber a quem deve-se enviar a resposta da movimentação, sua posição horizontal é necessária para que o robô mestre veja se o robô escravo se aproximou ou distanciou do mestre na formação, por fim, a posição vertical é necessária para que um robô não fique em uma posição à frente do outro, por exemplo, em uma formação em linha, a posição vertical deve ser a mesma para todos os robôs, logo, se um robô está em uma posição mais a frente, ele deve esperar até que os que estão mais atrás cheguem a essa mesma linha.

O pseudocódigo do processo de cooperação entre dois robôs pode ser visto na Figura 5.10. Este algoritmo é válido para a cooperação entre o robô mestre e o robô escravo à sua esquerda. A primeira condição verifica se os robôs se aproximaram. Além disso, a condição também verifica se o robô mestre se deslocou em direção ao robô escravo (neste caso, para à esquerda) ou se o robô escravo se deslocou em direção ao robô mestre (neste caso, para à direita). Também é necessário verificar se os robôs ainda não iniciaram a troca de posição entre eles. Se esse primeiro teste suceder, faz-se a troca de posições, onde hor_m receberá o valor de hor_e e vice-versa, além do escravo se tornar o mestre e o mestre se tornar escravo. Também se atualiza as variáveis que guardam se os robôs estão executando uma troca de posição. Essa será a única ação a ser aplicada nesse passo de tempo. Apenas no próximo passo de tempo, após os robôs terem trocado suas colunas de referência, ocorrerá uma movimentação de fato. Podemos ressaltar aqui a necessidade de verificar no primeiro teste se, além de se aproximarem, os robôs ainda não não estão trocando de posição entre eles. No instante de tempo após a troca, se o teste

fosse apenas da aproximação entre os robôs, novamente o teste sucederia pois os robôs ainda se encontram nas mesmas posições, apenas com a troca de papéis entre eles. Porém, sabendo que os robôs iniciaram a troca de posições, pode-se avançar para a segunda parte do pseudocódigo, na qual não ocorrerá a troca de posições e sim a aplicação de alguma regra de controle de formação mostrada na Figura 5.9. Dessa forma, a segunda parte do pseudocódigo será aplicada nas seguintes situações: (i) se os robôs se aproximaram, mas apenas após a troca; (ii) se os robôs não se aproximaram. Na segunda parte do algoritmo, caso os robôs estejam trocando de posição, ambos aplicarão o controle de formação para trocarem de coluna o mais rápido o possível, caso contrário, o objetivo é identificar se o par de robôs está alinhado, ou seja, se os dois robôs se encontram na mesma posição vertical (y). Basicamente, a ideia é que se os dois robôs estão alinhados, ambos serão deslocados; caso contrário, apenas o robô que está atrás dará um passo de deslocamento (o que faz com que o robô mais avançado aguarde a progressão daquele que está atrasado). Quando ocorrer, o deslocamento de um robô se dará de acordo com as regras da Figura 5.9. Ou seja, será um passo para a frente se o robô estiver na sua coluna original, ou um passo na diagonal superior, se ele estiver deslocado. No pseudocódigo da Figura 5.10, se o primeiro teste da aproximação falhar, o próximo teste verifica se ambos os robôs estão na mesma posição vertical (y_m=y_e), assim aplica-se a regra de formação para ambos; caso contrário, a regra de formação será aplicada apenas para o robô que está mais atrás (apenas o mestre se desloca, se y_m<y_e ou apenas o escravo, se y_e<y_m).

O pseudocódigo apresentado na Figura 5.10 é válido para a cooperação entre o robô mestre e o robô escravo à sua esquerda. Um pseudocódigo similar é aplicado para a cooperação entre o robô mestre e o robô escravo à sua direita, sendo que a única diferença está na primeira condição do algoritmo. Nesse caso, a primeira condição verifica se houve aproximação e se o robô mestre se deslocou para a direita, ou se o robô escravo se deslocou em direção ao robô mestre (para a esquerda). Assim, o teste para a primeira condição é: if ($|x_m$-$x_d| < dist$) & (x_m > hor(m) || x_d < hor(d)) & !(trocando_posicoes(m,e).

O restante do pseudocódigo é exatamente igual ao da Figura 5.10, apenas trocando "e" (escravo à esquerda) por "d" (escravo à direita).

A decisão entre aplicar o algoritmo primeiro ao par (mestre, escravo à esquerda) ou ao par (mestre, escravo à direita) se dá unicamente em função da ordem de chegada das mensagens até o robô mestre. Ou seja, se o mestre receber primeiro uma mensagem do robô escravo à direita com sua posição e uma solicitação de regra de formação a ser aplicada, o mestre irá efetuar primeiro o pseudocódigo desse par.

Os experimentos em [Ioannidis et al. 2011b] foram feitos tanto em simulação quanto com robôs reais. Para os experimentos reais, foram utilizados três robôs e-puck [École Polytechnique Fédérale de Lausanne EPFL 2013] e nenhum computador forneceu dados para o robô durante o processo de movimentação, visto que deseja-se um modelo onde os robôs ajam de forma autônoma. Para montar a vizinhança, os robôs utilizaram seus

```
//m = robo mestre
//e = robo escravo da esquerda
//Robôs se deslocaram em direção do outro
if |xm - xel < dist & (xm < hor(m) || xe > hor(e)) & !(trocando_posicoes(m,e)) then
    //Robôs se aproximaram
    troca_posicoes(m, e)
else
    //Robôs trocando de posição
    if trocando_posicoes(m, e) then
        aplica_formacao(m)
        aplica_formacao(e)
    else
        //Robôs estão na formação ou se distanciaram
        if ym == ye then
            aplica_formacao(m)
            aplica_formacao(e)
        else
            if ym < ye then
                aplica_formacao(m)
            else
                aplica_formacao(e)
            end-if
        end-if
    end-if
end-if
```

Figura 5.10: Pseudo-código do método de controle de formação do modelo de Ioannidis e colegas.

sensores de proximidade infra-vermelho, e para fazer a comunicação entre os robôs, foram trocadas mensagens via protocolo *Bluetooth*.

5.2 Análise do Modelo Original com Um Robô

O modelo discutido em [Ioannidis et al. 2008] foi implementado inicialmente em simulação para que fosse verificado o comportamento descrito nos artigos. O software de simulação escolhido foi o Webots [Cyberbotics 2013], que proporciona a simulação de ambientes que contêm robôs e-pucks [École Polytechnique Fédérale de Lausanne EPFL 2013], além de possibilitar que o código testado em simulação seja facilmente embarcado nos robôs reais. Entretanto, o modelo descrito no artigo original pressupõe um ambiente com um time de robôs que deve agir cooperativamente. Em um primeiro momento, esta característica foi modificada para que apenas um robô se movimentasse no ambiente, verificando-se o comportamento do modelo sem a presença da cooperação. Dessa forma, teríamos um cenário mais simples para o entendimento e a implementação do modelo na plataforma de simulação e em robôs reais. Além disso, essa abordagem possibilitou que o estudo inicial fosse mais focado no modelo em relação às regras de desvio de obstáculo.

Porém, para que fosse possível implementar o modelo com a presença de somente um robô, foi necessária a mudança do algoritmo de controle de formação. O algoritmo

original descrito na Seção 5.1.2 pressupõe a existência de pelo menos dois robôs no time, que trocariam de posição quando necessário, assim o robô mestre tomaria a posição do robô escravo e vice-versa. Como só existe um robô no ambiente, não existe a fase de troca de coluna na formação quando dois robôs se aproximam. Também não é necessário verificar o valor de y, pois só existe um robô. Assim, o pseudocódigo da Figura 5.10 é desnecessário para o cenário com um único robô e, no caso do robô se deslocar de sua coluna original, ele irá somente retornar à sua coluna original o mais rápido possível aplicando as regras de controle de formação já descritas na Figura 5.9.

Um exemplo de percurso do robô, de acordo com o modelo, alternando a aplicação da regra de desvio de obstáculos e da regra de controle de formação, pode ser visto na Figura 5.11. Na Figura 5.11.a, vemos o robô com um obstáculo à frente que o impede de se deslocar nessa direção. Logo, ele deve utilizar a regra de desvio de obstáculos. Se analisarmos a regra geral apresentada na Figura 5.8, veremos que nessa situação, onde existem obstáculos nas células a_1 e a_2 da vizinhança cuja célula central contém o robô e todas as outras vizinhas estão livres, o robô deve rotacionar para a esquerda (90°). Na Figura 5.11.b, vemos que o robô se encontra na mesma célula após aplicar uma rotação de 90°. No próximo passo, de acordo com a regra apresentada na Figura 5.8, o robô irá se deslocar na direção direita-esquerda, uma vez que não existem obstáculos na célula a_7. Na Figura 5.11.c, vemos o robô se deslocar para a célula à esquerda, sendo que o obstáculo agora está na célula a_2 da vizinhança do robô. Após aplicar novamente a regra de desvio de obstáculos, nessa situação o robô ainda manterá a orientação de 90°, se deslocando mais uma célula a esquerda atingindo a configuração da Figura 5.11.d. Nesse passo, temos o robô sem nenhum obstáculo em sua vizinhança, porém não está em sua orientação original e aplica uma regra de desvio de obstáculo apenas para rotacionar de volta a 0° (última regra da Figura 5.8). Assim, ele atinge a configuração da Figura 5.11.e, na qual não existem obstáculos na vizinhança do robô e o mesmo está na orientação original, podendo aplicar a regra de controle de formação. Aplicando as regras descritas na Figura 5.9, o robô se desloca para a célula diagonal superior direita, atingindo a nova posição mostrada na Figura 5.11.f. Ao se encontrar novamente ao lado do obstáculo (célula a_3), será aplicada a regra de desvio de obstáculo da Figura 5.8 e com isso o robô irá dar uma passo para o norte, se encontrando na posição mostrada na Figura 5.11.g. No próximo passo, a regra de desvio da Figura 5.8 será aplicada novamente (obstáculo na célula a_4): como as células a_1, a_2 e a_8 estão livres o robô dará mais um passo ao norte (Figura 5.11.h). No último passo, o robô não encontrará nenhum obstáculo em sua vizinhança novamente e irá aplicar uma regra de controle de formação que o fará se deslocar na diagonal, atingindo a configuração da Figura 5.11.i. Como nessa última configuração o robô atinge a célula meta, o robô termina a sua movimentação.

O modelo com um único robô foi então implementado no ambiente de simulação Webots, utilizando o robô e-puck como a arquitetura a ser simulada. A Figura 5.12 mostra

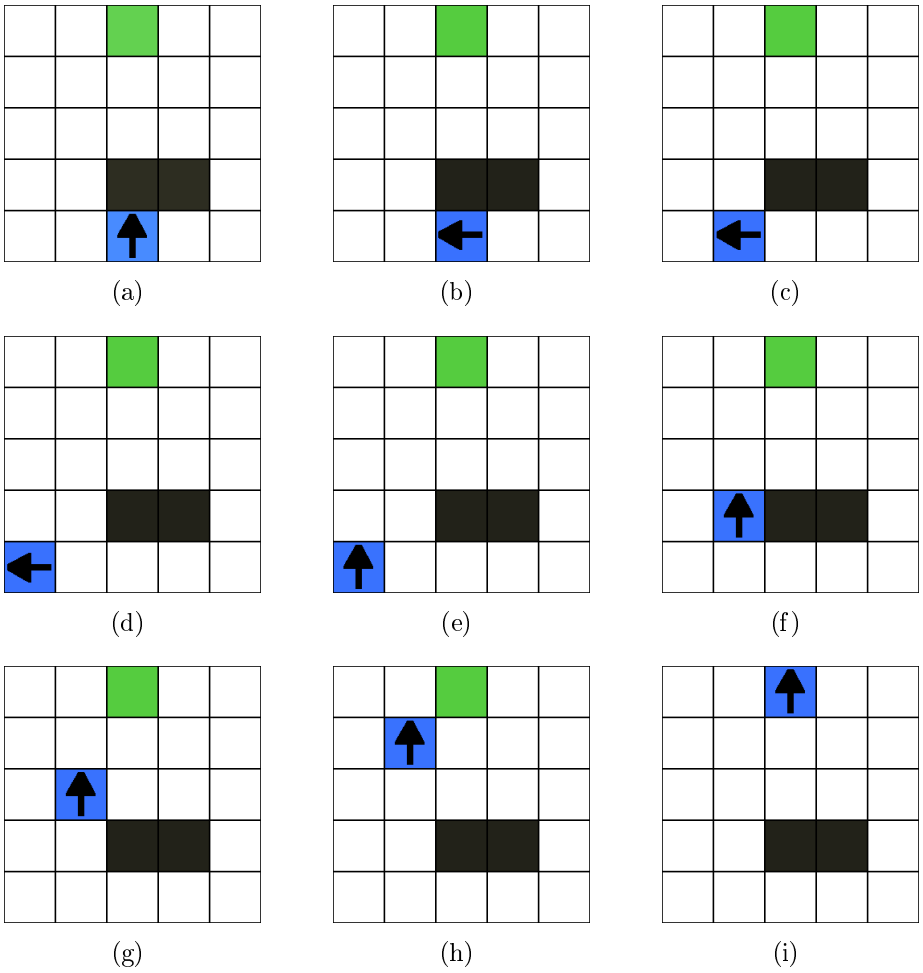


Figura 5.11: Exemplo de percurso em ambiente com um robô. Célula azul representa o robô, sendo a seta a indicação da orientação naquele passo de tempo, a célula verde a meta, e as células pretas obstáculos.

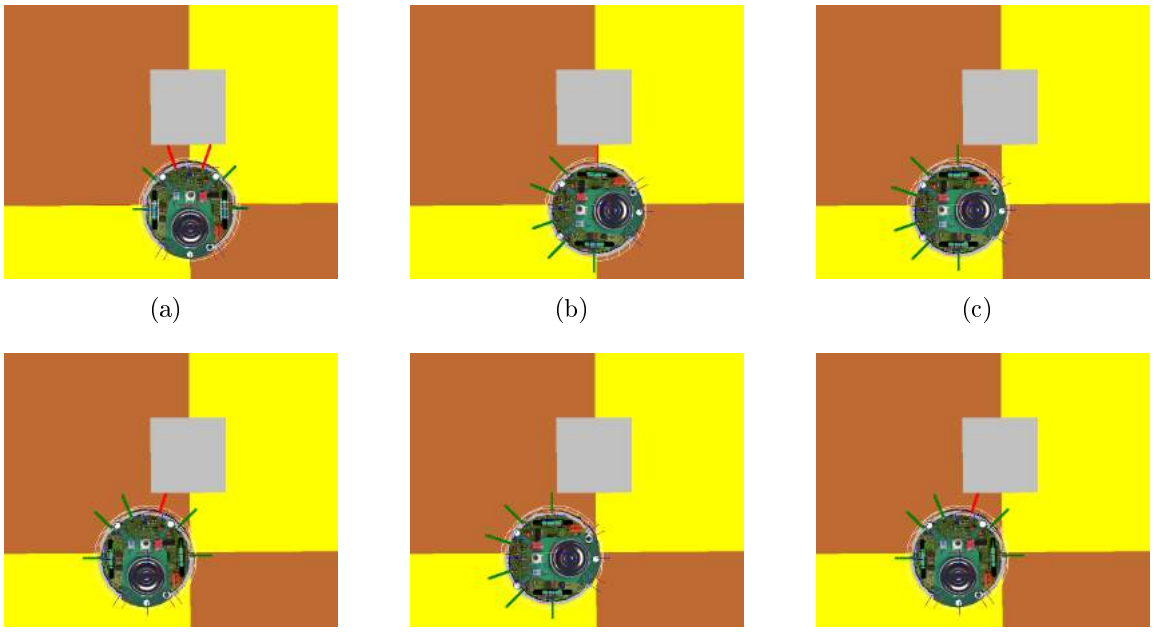
um exemplo de simulação gerada através do Webots, no qual procuramos reproduzir o cenário e o comportamento do robô exemplificado na Figura 5.6, utilizando 1 robô e-puck e 1 obstáculo. Ou seja, o robô deve se deslocar na direção sul-norte e encontra um obstáculo a frente, devendo desviar a esquerda para contorná-lo. Entretanto, nessa simulação foi observado um problema que impossibilitou o robô de encontrar a meta, que consiste em um "deadlock" na movimentação do robô, fazendo com que ele fique em estado de rotação contínua, sem voltar a se deslocar. Este problema foi identificado quando o robô está em uma quina de um obstáculo que ele está contornando. Por exemplo, na Figura 5.12, o robô se depara com o obstáculo a frente (Figura 5.12.a), realiza a rotação a 90° (Figura 5.12.b) e realiza seu deslocamento a esquerda da direção original (enquanto o obstáculo for identificado à direita pelos sensores do robô) até o cenário da Figura 5.12.c, no qual os sensores do robô não mais identificam a presença de obstáculo a direita. Nesse caso, de acordo com a regra de desvio de obstáculo, o robô deve rotacionar sobre seu eixo, voltando à orientação original de deslocamento (0°), como na Figura 5.12.d. Porém, ao voltar para a orientação original, os sensores da frente do robô voltam a identificar um

obstáculo a frente, pois de fato o obstáculo ainda não foi totalmente superado. De acordo com a regra de desvio de obstáculo, nesse cenário, o robô deverá fazer uma rotação 90° novamente para realizar o contorno do obstáculo à esquerda como na Figura 5.12.e, voltando à mesma configuração apresentada na Figura 5.12.c. Assim, novamente o obstáculo não é identificado pelos sensores laterais e o robô gira para voltar à orientação original como na Figura 5.12.f, repetindo a configuração da Figura 5.12.d. O robô voltará a repetir essas rotações indefinidamente, alternando entre as configurações das figuras 5.12.c. e 5.12.d. Chamamos esse problema de "Deadlock da Quina do Obstáculo".

Esse problema ocorre devido à diferença na precisão entre os sensores frontais e os laterais do robô e-puck. Observe que de fato na Figura 5.12.c, embora o robô tenha se deslocado e contornado uma boa parte do obstáculo, o mesmo ainda não foi totalmente superado e ainda se encontra à direita de aproximadamente 50% do corpo do robô e-puck. Entretanto, nessa situação os sensores laterais não são capazes de identificar a presença desse obstáculo (uma vez que o sensor lateral central já passou pela quina do obstáculo) e apresentam uma leitura próxima de 0. Por outro lado, quando o robô gira novamente e o obstáculo volta a estar a sua frente, os sensores frontais são capazes de identificar o obstáculo, forçando um novo giro do robô. Este ciclo se repete infinitamente.

Quando identificamos esse problema, retornamos aos artigos de Ioannidis e colaboradores para checar se existia algum relato relacionado a esse cenário, visto que os autores também empregaram a arquitetura e-puck em suas simulações e experimentos reais. Entretanto, não encontramos nenhuma menção a esta questão e nos experimentos apresentados os robôs parecem superar as quinas sem nenhum problema. No primeiro artigo [Ioannidis et al. 2008], os autores apresentam simulações e experimentos reais que envolvem um time de robôs que empregam as regras de controle de formação em linha reta. Por outro lado, as regras de desvio de obstáculo são as mesmas e acreditamos que tal situação também ocorreria. Mas, nos experimentos ilustrados no artigo, sempre que o robô chega em uma quina de obstáculo existe também uma aproximação entre esse robô e um outro da linha, fazendo com que as regras de controle de formação forcem o robô a se mover na diagonal e ele não tenta "contornar a quina", fazendo um cruzamento para a coluna a esquerda. Entretanto, embora nos exemplos apresentados no artigo essa regra de formação tenha auxiliado o robô com a troca de colunas, acreditamos que dependendo do cenário, o robô poderia chegar na quina e não realizar essa troca (por exemplo, se não houvesse aproximação com o robô a sua esquerda), o que levaria o robô a tentar retornar a sua coluna original e ele fatalmente cairia no cenário do *deadlock* que apresentamos, caso fossem empregadas as regras de desvio de obstáculo da forma descrita no artigo. No artigo posterior [Ioannidis et al. 2011a], os autores também não mencionam esse comportamento de *deadlock* nas quinas, mas relatam a utilização de uma arquitetura e-puck modificada com 36 sensores, para melhorar a identificação dos obstáculos. Acreditamos que com essa modificação no hardware, o robô não apresentaria esse problema pois prova-

velmente no cenário da Figura 5.12.c, os sensores seriam capazes de identificar o obstáculo a direita. Entretanto, na arquitetura original do e-puck, esse problema é crítico, conforme apresentado nas próprias simulações e depois confirmado com experimentos envolvendo equipamentos reais. Além disso, ainda que outras arquiteturas fossem utilizadas, é comum a existência de "pontos cegos" nas leituras de sensores, especialmente nas laterais. Assim, acreditamos que uma melhoria no modelo se faz necessária, para que o algoritmo possa reconhecer que o robô se encontra em uma situação de *deadlock* e tomar uma decisão diferente (no caso, continuar a contornar o obstáculo até superar a quina). A próxima seção apresenta uma modificação que efetuamos no modelo a fim de corrigir esse problema.



(a) (b) (c)

(d) (e) (f)

Figura 5.12: Exemplo de caso onde ocorre o *deadlock* na movimentação do robô.

5.3 Novo Modelo Baseado na Regra de Atualização Local para cenários com 1 robô

A partir das observações iniciais a respeito do comportamento do robô no modelo descrito em [Ioannidis et al. 2008], em especial após a identificação do problema do *deadlock* na quina do obstáculo descrita no exemplo da Figura 5.12, realizamos algumas modificações de forma a melhorar seu comportamento. Esse modelo preserva as características principais do apresentado em [Ioannidis et al. 2008], ou seja: (a) utiliza um modelo baseado na regra de atualização local de um AC para decidir o passo do robô a cada passo de tempo; (b) essa regra de atualização é formada por dois conjuntos de regras locais, sendo que o primeiro é utilizado em situações de desvio de obstáculos e o segundo é utilizado para manter uma formação desejada durante a navegação do robô pelo ambiente. Nas

primeiras modificações que realizamos, que são descritas nessa seção, foram considerados apenas cenários com um único robô e as regras de controle de formação apenas forçam o robô a retornar a sua coluna original, após desviar-se de um obstáculo.

5.3.1 Primeira Alteração: Inserção do estado "Robô Rotacionado" para correção do *deadlock* de quina do obstáculo

A primeira alteração no modelo visa resolver o problema identificado como "deadlock de quina do obstáculo" descrito na seção anterior. Para isso, a ideia principal é utilizar um estado diferenciado para o modelo ser capaz de identificar que o robô já vinha de uma rotação para desvio de obstáculo ao fazer uma nova rotação e voltar para a orientação original identificando novamente um obstáculo. Assim, quando o robô chega numa situação em que o obstáculo que está sendo desviado não é mais identificado pelos sensores laterais, ao realizar a rotação de volta para a orientação original como o cenário da Figura 5.12.d, o robô altera seu estado para "Robô_Rotacionado". Caso o robô realmente não identifique nenhum obstáculo a frente, ele voltará a se deslocar para a frente, voltando ao estado "Robô" e executando os passos normais após o desvio completo de um obstáculo. Por outro lado, caso o estado seja "Robô_Rotacionado" e os sensores voltem a sinalizar um obstáculo na célula a_7, o modelo identifica uma situação de "quina do obstáculo" e faz o robô rotacionar novamente para desviar do obstáculo caminhando mais uma célula à esquerda, ainda que a leitura dos sensores laterais não indique a presença do obstáculo. A ideia é que ao se identificar um obstáculo vindo de uma rotação que só ocorreria na ausência de obstáculos, o modelo identifique que o robô está diante de um "ponto cego" e rotacione novamente a 90° ignorando a leitura do sensor para dar mais um passo a esquerda, o que não aconteceria se o estado fosse "Robô". Assim, a primeira mudança no modelo refere-se à inclusão de mais um estado possível por célula chamado de "Robô_Rotacionado" e a alteração das regras de desvio de obstáculo para trabalharem com esse novo estado. A Figura 5.13 mostra o novo conjunto de regras de desvio de obstáculos utilizado nesta modificação.

Comparando-se essas regras da Figura 5.13 com as regras originais apresentadas na Figura 5.8, é possível perceber que a transição referente à vizinhança que contém obstáculo na célula central permanece inalterada. Porém as transições referentes às vizinhanças que contêm a célula central livre ou com a presença do robô devem ser adaptadas para suportar a adição do novo estado Robô_Rotacionado, além de ser necessário incluir novas transições para o caso em que o robô está na célula central da vizinhança no estado Robô_Rotacionado. A seguir, descrevemos as principais modificações:

- Nas vizinhanças com a célula central livre: (i) se o robô estiver em 0° posicionado na célula a_5 independentemente se o estado da célula a_5 é Robô ou Robô_Rotacionado, o robô será deslocado para a célula central, sendo seu novo estado Robô; (ii) se o robô

Célula Central	Ângulo	Vizinhança	Nova Célula Central	Novo ângulo
LIVRE	0°	(a5 = ROBO OR ROBO_ROTACIONADO) & a3 & a7 = LIVRE	ROBO	0°
		Caso Contrário	LIVRE	0°
	-90°	a3=ROBO & a5 = LIVRE & a2 = OBSTACULO	ROBO	-90°
		a3 = ROBO_ROTACIONADO	ROBO	-90°
		Caso Contrário	LIVRE	-90°
	90°	a7=ROBO & a5 = LIVRE & a8 = OBSTACULO	ROBO	90°
		a7 = ROBO_ROTACIONADO	ROBO	90°
		Caso Contrário	LIVRE	90°
OBSTACULO	Qualquer	Qualquer Vizinhança	OBSTACULO	Qualquer
ROBO	0°	a1 & a2 & a8 = LIVRE	LIVRE	0°
		a1 = OBSTACULO & a2 & a3 = LIVRE	ROBO	-90°
		a1 & (a2 OR a3) = OBSTACULO & a7 & a8 = LIVRE	ROBO	90°
		a1 & a7 = OBSTACULO & a3 = LIVRE	ROBO	-90°
		a1 & a3 = OBSTACULO & a7= LIVRE	ROBO	90°
		a8 = OBSTACULO & a3 = LIVRE	ROBO	-90°
		a2 = OBSTACULO & a7 = LIVRE	ROBO	90°
	-90°	a1 = OBSTACULO & a7 & a6 = LIVRE	LIVRE	-90°
		a1 = LIVRE	ROBO_ROTACIONADO	0°
	90°	a1 = OBSTACULO & a3 & a4 = LIVRE	LIVRE	90°
		a1 = LIVRE	ROBO_ROTACIONADO	0°
ROBO_ROTACIONADO	0°	a1 & a2 & a8 = LIVRE	LIVRE	0°
		a1 = OBSTACULO & a2 & a3 = LIVRE	ROBO_ROTACIONADO	-90°
		a1 & (a2 OR a3) = OBSTACULO & a7 & a8 = LIVRE	ROBO_ROTACIONADO	90°
		a1 & a7 = OBSTACULO & a3 = LIVRE	ROBO_ROTACIONADO	-90°
		a1 & a3 = OBSTACULO & a7= LIVRE	ROBO_ROTACIONADO	90°
		a8 = OBSTACULO & a3 = LIVRE	ROBO_ROTACIONADO	-90°
		a2 = OBSTACULO & a7 = LIVRE	ROBO_ROTACIONADO	90°
	-90°	a7 = LIVRE	LIVRE	-90°
		Caso Contrário (Não há caminho)	ROBO_ROTACIONADO	-90°
	90°	a3 = LIVRE	LIVRE	90°
		Caso Contrário (Não há caminho)	ROBO_ROTACIONADO	90°

Figura 5.13: Regras de desvio de obstáculo para a primeira modificação do modelo de Ioannidis e colegas.

estiver em -90° posicionado na célula a_3, deve-se diferenciar se o estado da célula a_3 é Robô ou Robô_Rotacionado: (ii.a) se for o estado Robô_Rotacionado, significa que o robô dará um passo a mais para a esquerda, para escapar da quina do obstáculo, voltando ao estado Robô; (ii.b) se for o estado Robô deve-se analisar se a célula a_2 possui um obstáculo; se possuir ele dará mais um passo a esquerda mantendo o estado Robô; (ii.c) em todos os outros casos, a célula central permanece livre; (iii) se o robô estiver em 90° posicionado na célula a_7, as transições são similares ao descrito anteriormente para o robô a -90° posicionado na célula a_3, apenas modificando-se que o robô desloca para a direita.

- Nas vizinhanças com o robô posicionado na célula central e no estado Robô: (i) se o ângulo atual for 0°, as transições permanecem iguais ao do modelo original (Figura 5.8); (ii) se o ângulo atual for -90° e a célula a_1 estiver livre, a transição estabelece que o novo ângulo do robô é 0°, disparando uma rotação do robô, e também modifica o novo estado da célula central para Robô_Rotacionado, indicando que o robô irá retornar para a orientação 0° vindo de uma rotação anterior. (iii) se o ângulo atual for 90°, a transição é modificada da mesma forma descrita anteriormente para o ângulo de -90°.
- Nas vizinhanças com o robô posicionado na célula central e no estado Robô_Rotacionado: (i) se o robô está em -90° (ou 90°), e a células a_7 (a_3) é livre, o robô deve dar mais um passo para escapar da quina do obstáculo, tornando a célula central livre e mantendo o ângulo de orientação do robô; (ii) se o robô está em -90° (ou 90°), e a célula a_7 (a_3) não é livre, não é possível o robô escapar desta quina, pois encontrou um obstáculo enquanto tentava escapar, assim ele permanece parado; (iii) se o robô está em 0°, as regras são similares àquelas quando o estado da célula central é igual a Robô, apenas alterando-se que o estado da célula central se mantém como Robô_Rotacionado.

Após a implementações dessas modificações nas regras de desvio de obstáculos, o novo modelo foi simulado na plataforma Webots. A Figura 5.14 apresenta o resultado de uma simulação utilizando o mesmo cenário inicial da Figura 5.12 no qual o deadlock havia sido observado. A Figura 5.14.a mostra o robô com um obstáculo à frente, ele então rotaciona para o ângulo de 90°, como mostrado na Figura 5.14.b, e se desloca a esquerda para sair do obstáculo. Quando encontra a posição mostrada na Figura 5.14.c, ele não identifica mais o obstáculo com seu sensor lateral e retorna à 0°, porém alterando o estado da célula central para Robô-Rotacionado, como mostrado na Figura 5.14.d. Na orientação original, os sensores frontais identificam o obstáculo, fazendo com que ele retorne ao ângulo 90°, como mostrado na Figura 5.14.e. Novamente os sensores laterais não são capazes de identificar o obstáculo. Entretanto, o estado da célula central é Robô-Rotacionado e, de acordo com a nova regra de desvio, o robô dará mais um passo para escapar da quina do obstáculo, ainda que nenhum obstáculo seja identificado, como mostrado na Figura 5.14.f. As figuras 5.14.g, 5.14.h e 5.14.i mostram que todo o processo se repete por mais uma vez, com o uso do estado Robô-Rotacionado novamente, para que o robô dê mais um passo para se afastar da quina. Finalmente, na Figura 5.14.j, o robô faz um novo giro para a orientação original e não encontrando o obstáculo a frente, consegue aplicar a regra de controle de formação, movimentando para a célula em direção diagonal superior direita. O processo de giro, movimentação e retorno para a orientação original pode ser visto nas Figuras 5.14.k, 5.14.l e 5.14.m. Na Figura 5.14, todas as configurações que foram alcançadas com o robô estando no estado Robô_Rotacionado na célula central da

sua vizinhança foram destacadas com um (*). Em todas as outras, a célula central da vizinhança do robô está no estado Robô.

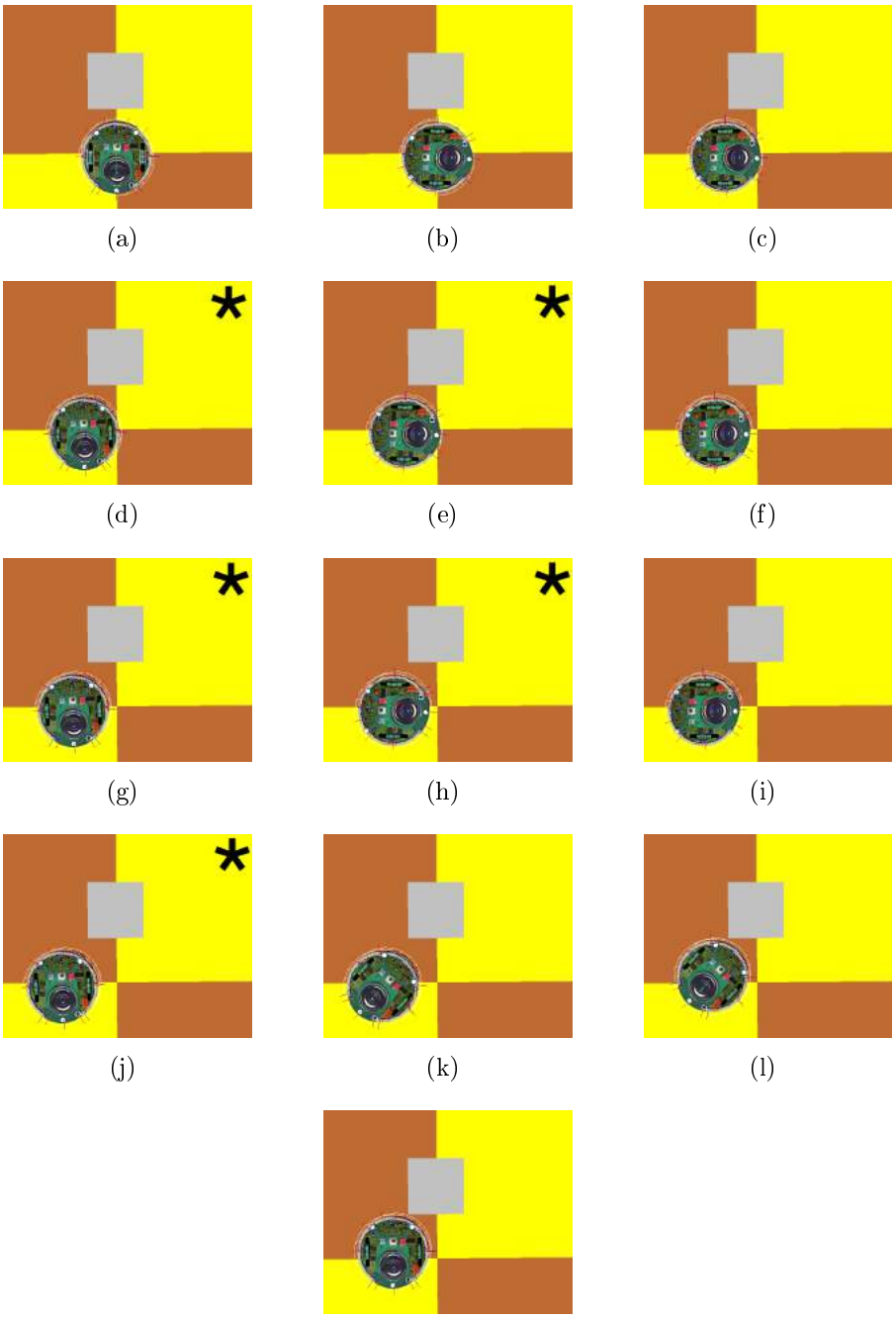


Figura 5.14: Exemplo de percurso com a correção do *deadlock*.

Assim, é possível perceber pela simulação da Figura 5.14.m que o problema do *deadlock* da quina foi superado com a introdução do estado Robô-Rotacionado, e o robô consegue se deslocar o suficiente para desviar-se totalmente do obstáculo e voltar a progredir na direção sul-norte. Entretanto, é possível observar que enquanto se desviava da quina o robô faz duas rotações para cada passo a esquerda, o que torna o processo lento. Assim, uma nova modificação foi realizada conforme descrevemos na próxima seção.

5.3.2 Segunda Alteração: Inserção de estados "Robô_Rotacionado_i" para um deslocamento maior na quina do obstáculo

Após os testes com a primeira alteração proposta, mostrada na Seção 5.3.1, verificou-se que mesmo sendo possível o robô desviar corretamente de um obstáculo no novo modelo, o processo de desvio da quina dos obstáculos ainda estava bastante lento. A cada retorno de uma rotação, o robô dava somente um passo para sair da quina do obstáculo, assim quando voltava ao ângulo 0°, os sensores frontais continuavam a encontrar o obstáculo à frente, sendo necessário uma nova rotação a 90°, para que o robô se deslocasse por mais um passo. Percebendo-se esse comportamento, foi proposta uma nova alteração que consistiu em aumentar a quantidade de estados que indiquem a situação de robô previamente rotacionado, para que o robô dê mais passos para desviar de uma quina de obstáculo antes de retornar ao ângulo 0°.

A quantidade de passos definida no novo modelo para que o robô escape totalmente da quina do obstáculo levou em conta o raio do robô e-puck [École Polytechnique Fédérale de Lausanne EPFL 2013], que é aproximadamente 3.5 centímetros. Como o tamanho de cada célula utilizado em nossos experimentos é igual à 1 centímetro, seria necessário no mínimo quatro passos, ou quatro células, para o robô ultrapassar completamente a quina, ou seja, para que a sua metade traseira passasse completamente do obstáculo. Assim, em nossa descrição do novo modelo iremos apresentar uma alteração para que sempre que o robô identifique a situação que já foi rotacionado e encontre novamente um obstáculo à frente, ele volte a se deslocar para a esquerda por 4 passos consecutivos. Entretanto, no modelo esse número de passos pode ser um parâmetro a ser ajustado de acordo com o tamanho das células e das dimensões do robô.

Nesta alteração, quatro novos estados foram adicionados: "Robô_Rotacionado_0", "Robô_Rotacionado_1", "Robô_Rotacionado_2", "Robô_Rotacionado_3". O índice i de cada estado "Robô_Rotacionado_i" indica que o robô já se deslocou uma quantidade de passos i para desviar de uma quina de obstáculo, após a identificação da mesma (ou seja, na primeira rotação que identificou um obstáculo a frente). O novo conjunto de regras para atender à esta alteração é mostrado na Figura 5.15. Comparando-se com a primeira alteração, apresentada na Figura 5.13, temos que as únicas mudanças são as transições que antes iam para o estado Robô_Rotacionado agora vão para o estado Robô_Rotacionado_0, e as transições que continham células livres como célula central e que irão receber algum Robô_Rotacionado_i, irão fazer o incremento do índice i a cada novo passo que ele caminhar, ou seja, o Robô_Rotacionado_0 irá se transformar em Robô_Rotacionado_1 após se deslocar 1 passo a esquerda, o Robô_Rotacionado_1 irá se transformar em Robô_Rotacionado_2 após o segundo passo a esquerda, até o estado Robô_Rotacionado_3 que irá se transformar no estado Robô após o quarto passo a esquerda. Todas as transições da Figura 5.15 para as vizinhanças com a cé-

lula central igual a Robô_Rotacionado devem ser repetidas para os quatro novos estados Robô_Rotacionado_0, Robô_Rotacionado_1, Robô_Rotacionado_2, Robô_Rotacionado_3. Na Figura 5.15, por simplificação, elas foram representadas de forma esquemática para o estado genérico Robô_Rotacionado_i. Portanto, essas regras são multiplicadas por 4, em relação à versão anterior do modelo descrita na Figura 5.13. Todas as outras transições permanecem idênticas à versão anterior.

Célula Central	Ângulo	Vizinhança	Nova Célula Central	Novo Ângulo
LIVRE	0°	(a5 = ROBO OR ROBO_ROTACIONADO_0) & a3 & a7 = LIVRE	ROBO	0°
		Caso Contrário	LIVRE	0°
	-90°	a3=ROBO & a5 = LIVRE & a2 = OBSTACULO	ROBO	-90°
		a3 = ROBO_ROTACIONADO_0	ROBO_ROTACIONADO_1	-90°
		a3 = ROBO_ROTACIONADO_1	ROBO_ROTACIONADO_2	-90°
		a3 = ROBO_ROTACIONADO_2	ROBO_ROTACIONADO_3	-90°
		a3 = ROBO_ROTACIONADO_3	ROBO	-90°
		Caso Contrário	LIVRE	-90°
	90°	a7=ROBO & a5 = LIVRE & a8 = OBSTACULO	ROBO	90°
		a7 = ROBO_ROTACIONADO_0	ROBO_ROTACIONADO_1	90°
		a7 = ROBO_ROTACIONADO_1	ROBO_ROTACIONADO_2	90°
		a7 = ROBO_ROTACIONADO_2	ROBO_ROTACIONADO_3	90°
		a7 = ROBO_ROTACIONADO_3	ROBO	90°
		Caso Contrário	LIVRE	90°
OBSTACULO	Qualquer	Qualquer Vizinhança	OBSTACULO	Qualquer
ROBO	0°	a1 & a2 & a8 = LIVRE	LIVRE	0°
		a1 = OBSTACULO & a2 & a3 = LIVRE	ROBO	-90°
		a1 & (a2 OR a3) = OBSTACULO & a7 & a8 = LIVRE	ROBO	90°
		a1 & a7 = OBSTACULO & a3 = LIVRE	ROBO	-90°
		a1 & a3 = OBSTACULO & a7= LIVRE	ROBO	90°
		a8 = OBSTACULO & a3 = LIVRE	ROBO	-90°
		a2 = OBSTACULO & a7 = LIVRE	ROBO	90°
	-90°	a1 = OBSTACULO & a7 & a6 = LIVRE	LIVRE	-90°
		a1 = LIVRE	ROBO_ROTACIONADO_0	0°
	90°	a1 = OBSTACULO & a3 & a4 = LIVRE	LIVRE	90°
		a1 = LIVRE	ROBO_ROTACIONADO_0	0°
ROBO_ROTACIONADO_i	0°	a1 & a2 & a8 = LIVRE	LIVRE	0°
		a1 = OBSTACULO & a2 & a3 = LIVRE	ROBO_ROTACIONADO_i	-90°
		a1 & (a2 OR a3) = OBSTACULO & a7 & a8 = LIVRE	ROBO_ROTACIONADO_i	90°
		a1 & a7 = OBSTACULO & a3 = LIVRE	ROBO_ROTACIONADO_i	-90°
		a1 & a3 = OBSTACULO & a7= LIVRE	ROBO_ROTACIONADO_i	90°
		a8 = OBSTACULO & a3 = LIVRE	ROBO_ROTACIONADO_i	-90°
		a2 = OBSTACULO & a7 = LIVRE	ROBO_ROTACIONADO_i	90°
	-90°	a7 = LIVRE	LIVRE	-90°
		Caso Contrário (Não há caminho)	ROBO_ROTACIONADO_i	-90°
	90°	a3 = LIVRE	LIVRE	90°
		Caso Contrário (Não há caminho)	ROBO_ROTACIONADO_i	90°

Figura 5.15: Regras de desvio de obstáculo para a segunda modificação do modelo de Ioannidis e colegas.

Após a implementação dessas modificações nas regras de desvio de obstáculos, o novo modelo foi simulado na plataforma Webots. A Figura 5.16 apresenta o resultado de uma simulação utilizando o mesmo cenário inicial da Figura 5.12 no qual o deadlock foi observado e da Figura 5.14 onde apenas um estado Robô_Rotacionado foi utilizado. Até o passo relativo à Figura 5.16.f, o comportamento é exatamente o mesmo da Figura 5.14, ou seja, até o passo de deslocamento a esquerda no qual os sensores laterais deixam de identificar o obstáculo. Já a Figura 5.16.g, mostra o robô saindo do estado Robô-Rotacionado-1 e dando mais um passo em 90° e atualizando o seu estado para Robô-Rotacionado-2. A Figura 5.16.h consiste no terceiro passo do robô para desviar do obstáculo e por fim, a Figura 5.16.i mostra o quarto e último passo para escapar do obstáculo, onde o robô consegue sair completamente da quina e retornar o estado da célula para o estado Robô. A Figura 5.16.j, mostra o robô retornando para o ângulo 0° após não encontrar nenhum obstáculo na célula a_1. As figuras 5.16.k, 5.16.l e 5.16.m mostram o robô aplicando a regra de controle de formação para retornar à sua coluna original o mais rápido o possível. Na Figura 5.16, todas as configurações que foram alcançadas com o robô estando no estado Robô_Rotacionado_i na célula central da sua vizinhança foram destacadas com um $(*\,i)$. Em todas as outras, o robô está na célula central de sua vizinhança no estado Robô.

Assim, é possível perceber pelo exemplo da Figura 5.16 que com a inclusão de novos estados ao modelo, o robô apresentou um comportamento mais eficiente após a identificação da quina do obstáculo, uma vez que o número de rotações desnecessárias foi reduzido. A próxima seção apresentará uma última modificação efetuada para cenários com 1 robô, visando também um comportamento mais eficiente no momento em que o robô começa a retornar à sua coluna original.

5.3.3 Terceira Alteração: Inserção do Estado "Robô_Alinhado" nas regras de controle de formação para diminuição do "ziguezague" após o desvio de um obstáculo

Após as alterações mostradas anteriormente, o robô conseguiu desviar de um obstáculo completamente, além de ter tido um comportamento mais interessante, com menos rotações, para atravessar a quina de um obstáculo. Após o robô conseguir voltar a se deslocar para a frente, depois do contorno completo do obstáculo, outro ponto identificado como passível de melhoria em seu comportamento foi o deslocamento do mesmo na lateral do obstáculo, quando o robô está sujeito à regra de controle de formação que força o seu retorno à sua coluna original.

Para o retorno à posição inicial, foi verificado que o robô executa vários movimentos de "ziguezague" para caminhar na lateral de um obstáculo. Após o robô atravessar horizontalmente o obstáculo por completo, ele aplica as regras de controle de formação. No controle de formação, o robô tenta voltar o mais rápido o possível para sua coluna inicial,

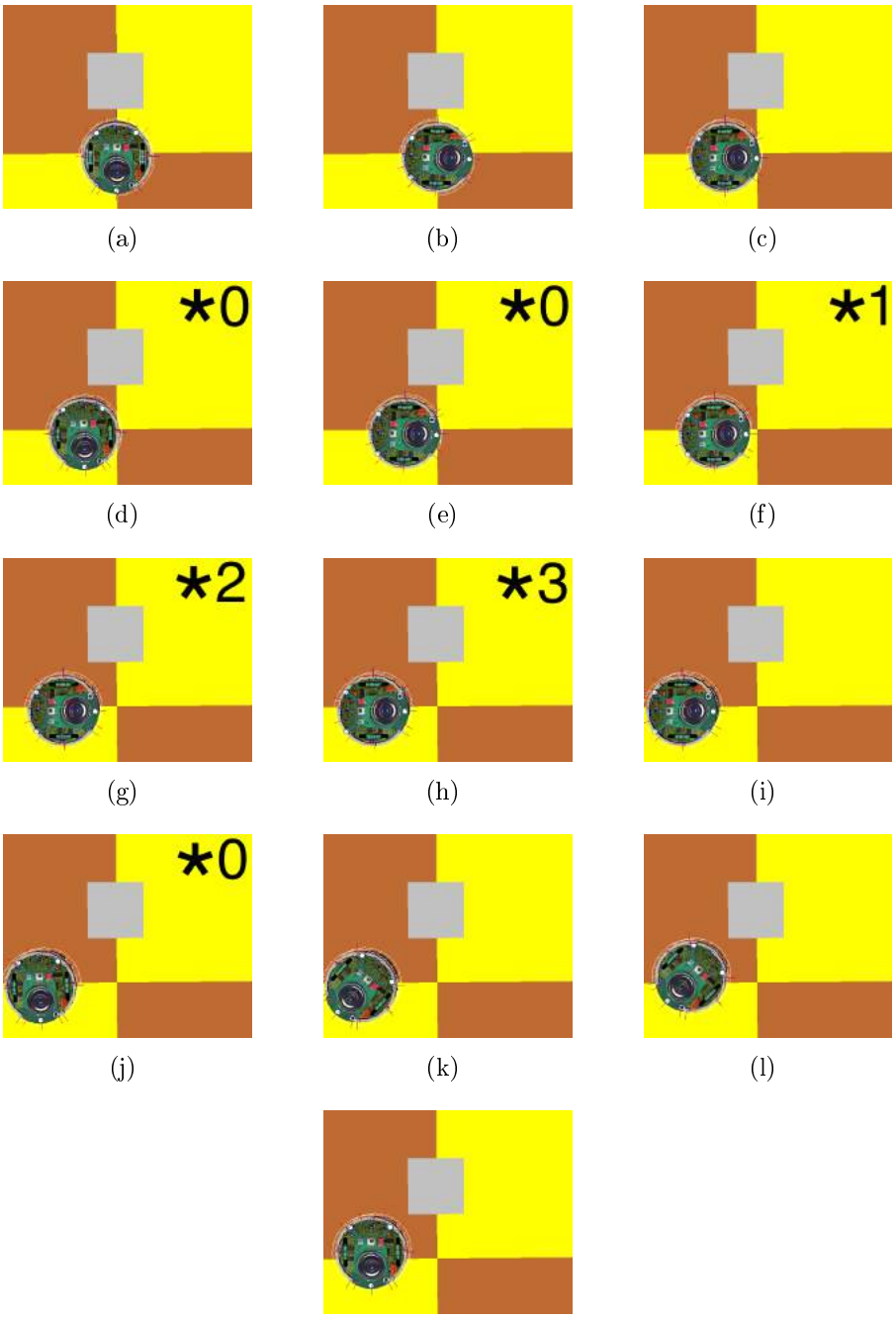


Figura 5.16: Exemplo de percurso com a segunda modificação do modelo.

através de movimentos na diagonal superior. Entretanto, logo após conseguir vencer a quina do obstáculo no movimento horizontal, é bastante provável que o robô ainda precise se deslocar verticalmente para superar a profundidade do obstáculo. Assim, nessa tentativa de voltar rapidamente a sua coluna inicial, o robô tende a se aproximar novamente do obstáculo, identificando-o através de seus sensores laterais. Nesse movimento, muitas vezes o robô acaba girando novamente à $90°$ (ou $-90°$) para se afastar do obstáculo. Depois de se afastar, ele volta a caminhar novamente em diagonal se aproximando do obstáculo e pode ficar nesse "bate e volta" na lateral por diversas vezes, dependendo da profundidade

do obstáculo. Assim, embora ele consiga progredir verticalmente durante os passos que dá na diagonal superior, os repetidos movimentos laterais acabam sendo um comportamento indesejado na movimentação do robô. Chamamos esse comportamento de deslocamento ziguezague do robô.

A Figura 5.17 exemplifica o comportamento de ziguezague. Na Figura 5.17.1, repetiu-se a Figura 5.16.j que ilustrou o último passo do robô no desvio da quina do obstáculo. A partir desse cenário, tem-se a movimentação apresentada na Figura 5.17. As figuras 5.17.2, 5.17.3 e 5.17.4 mostram a aplicação da regra de controle de formação que faz o robô se deslocar na diagonal superior direita. Ou seja, na Figura 5.17.1, não existem obstáculos na vizinhança e a regra de controle de formação é aplicada. Como nesse caso o robô se deslocou para a esquerda de sua coluna temos (hor_r- $x_r > 0$). Assim, de acordo com as regras da Figura 5.8, o robô deverá se deslocar 1 passo na direção da célula a_2 de sua vizinhança. Conforme explicado na Seção 5.1, esse deslocamento é composto na verdade por três movimentos: primeiro, o robô gira $-45°$ (Figura 5.17.2), depois o robô anda o equivalente a distancia de 1 célula na diagonal (Figura 5.17.3) e finalmente ele gira de volta 45° retornando a sua orientação original (norte), como mostra a Figura 5.17.4. A partir desse ponto, na ausência de obstáculos, a regra de controle de formação é novamente aplicada (5.17.5, 5.17.6 e 5.17.7) e o robô dá mais um passo na diagonal, aproximando-o ainda mais do obstáculo. Quando se encontra no cenário da Figura 5.17.7, o robô encontra um obstáculo à frente o que faz ele rotacionar novamente à 90° (Figura 5.17.8). Nesse cenário, como a célula a_1 não contém obstáculo, o robô retorna novamente ao ângulo 0°, porém alterando o estado da célula para Robô_Rotacionado_0, como mostrado na Figura 5.17.9. As figuras 5.17.10, 5.17.11, 5.17.12, 5.17.13 e 5.17.14 mostram o robô novamente aplicando as regras de desvio de obstáculo para desviar da quina, deslocando-se pelos 4 passos previstos na alteração do modelo descrita na Seção 5.3.2. A partir da Figura 5.17.15, o robô volta a aplicar regras de controle de formação para retornar à sua coluna original o mais rápido o possível, deslocando-se por 2 passos na diagonal até a Figura 5.17.24. Nesse cenário, o robô novamente identifica um obstáculo à frente rotacionando mais uma vez para 90°, como mostra a Figura 5.17.25. Da Figura 5.17.26 até a Figura 5.17.32, os movimentos são similares aos efetuados da Figura 5.17.9 até a Figura 5.17.15, sendo que o robô se desloca 4 passos a esquerda do obstáculo. A partir da Figura 5.17.33 até e 5.17.38, o robô aplica dois movimentos em diagonal. Quando chega na posição 5.17.38, o robô encontra um obstáculo na lateral, aplicando assim a regra de desvio de obstáculo que faz o robô ir pra frente, não sendo necessário retornar novamente mais 4 passos para a esquerda. Assim, nesse exemplo relativamente simples, o robô precisa alternar o deslocamento em diagonal (2 passos) com o afastamento a esquerda (quatro passos) por três vezes para que consiga superar completamente o obstáculo, evidenciando assim o ziguezague.

Para correção (ou atenuação) desse comportamento, nossa ideia básica foi fazer com

que o robô se movimentasse primeiramente um passo a frente antes de se deslocar na diagonal, sempre que ele estiver alinhado na orientação 0°, sem identificar qualquer obstáculo na vizinhança. Assim, um novo estado somente para o controle de formação foi adicionado ao modelo chamado Robo_Alinhado, não modificando o conjunto de regras de desvio de obstáculo definido nas seções anteriores. Sempre que o robô termina de desviar de um obstáculo, ele sai das regras de desvio de obstáculo no estado Robô com a orientação de 0°. A alteração implementada nas regras de controle de formação faz com que sempre que esteja no estado Robô, sem nenhum obstáculo na vizinhança, o robô deverá dar um passo a frente, independentemente de sua coluna atual. Entretanto, caso o robô esteja deslocado de sua coluna, após caminhar 1 passo a frente, ele irá mudar para o estado Robô_Alinhado. No estado Robô_Alinhado, ele volta a diferenciar seu comportamento de acordo com a coluna atual se deslocando para a diagonal superior direita (ou esquerda) e volta ao estado Robô. Assim, no controle de formação, caso o robô esteja deslocado de sua coluna original, ele passa a se deslocar em direção a ela, alternando entre 1 passo a frente e 1 passo na diagonal (e alternando entre o estado Robô e Robô_Alinhado). Assim que volta a sua coluna original, o robô passa a se deslocar apenas para frente, utilizando o estado Robô, até que volte a se deparar com um novo obstáculo. Caso o obstáculo seja identificado apenas pelos sensores na lateral do robô, ele irá se deslocar para a frente até terminar o obstáculo. Sendo Robô_A a representação do novo estado Robô_Alinhado para o controle de formação, o novo conjunto de regras para o controle de formação é mostrado na Figura 5.18.

Após as adaptações nas regras de controle de formação, foi executado o mesmo exemplo da Figura 5.16, onde os resultados serão mostrados na Figura 5.19.

A fim de comparação com o modelo sem as alterações, a Figura 5.19 também se inicia da mesma posição que a Figura 5.17. É possível ver que a Figura 5.19.b já é distinta com relação à 5.17.b, pois o robô dá um passo para frente antes de tentar retornar à sua coluna original. Nesse cenário, o robô está no estado Robô_Alinhado, que é representado na figura pelo símbolo "&". O passo na diagonal é ilustrado pelas figuras 5.19.c, 5.19.d e 5.19.e. Na Figura 5.19.f, o robô se depara com um obstáculo à frente, tendo que aplicar novamente a regra de desvio de obstáculos, dando os 4 passos para a esquerda (Figura 5.19.g a 5.19.m). Assim, na Figura 5.19.m, o robô novamente aplica a regra de controle de formação, dando primeiro um passo para frente e alterando o estado da célula para Robô_Alinhado e indo para a posição mostrada na Figura 5.19.n. A regra de controle de formação continua sendo aplicada até a Figura 5.19.u, alternando entre 1 passo para frente e 1 passo na lateral. No cenário da Figura 5.19.u o robô verifica que existe um obstáculo na lateral (mas não a frente) e aplica a regra de desvio de obstáculos, movimentando na direção sul-norte até encontrar a quina superior do obstáculo na Figura 5.19.y. A partir desse ponto, o robô não mais identifica obstáculos e passa a alternar 1 passo a frente com 1 passo na diagonal até alcançar sua coluna original.

Comparando-se o comportamento do robô nas simulações das figuras 5.17 e 5.19, é possível perceber que o robô diminuiu de 3 movimentos em ziguezague para apenas 1, ultrapassando o obstáculo e alcançando a coluna original de forma bem mais eficiente.

Com esta modificação, em alguns casos o robô demora mais para retornar à sua coluna inicial, visto que para cada passo em direção da coluna original, ele dá antes um passo para o norte. Porém, na maioria dos casos, principalmente quando não utilizada a segunda modificação apresentada na Seção 5.3.2, o tempo gasto fazendo o ziguezague é maior do que o tempo gasto para retornar à sua coluna inicial dando mais um passo para o norte antes de andar na diagonal. A título de comparação, para o primeiro exemplo, (Figura 5.17) foram necessários 39 passos para que o robô encontrasse um obstáculo na lateral e pudesse aplicar novamente a regra de desvio de obstáculo que faz o robô ir para a frente. No exemplo da Figura 5.19, em 22 passos o robô se encontra na mesma posição do final do primeiro exemplo, comprovando assim que o ziguezague é diminuído.

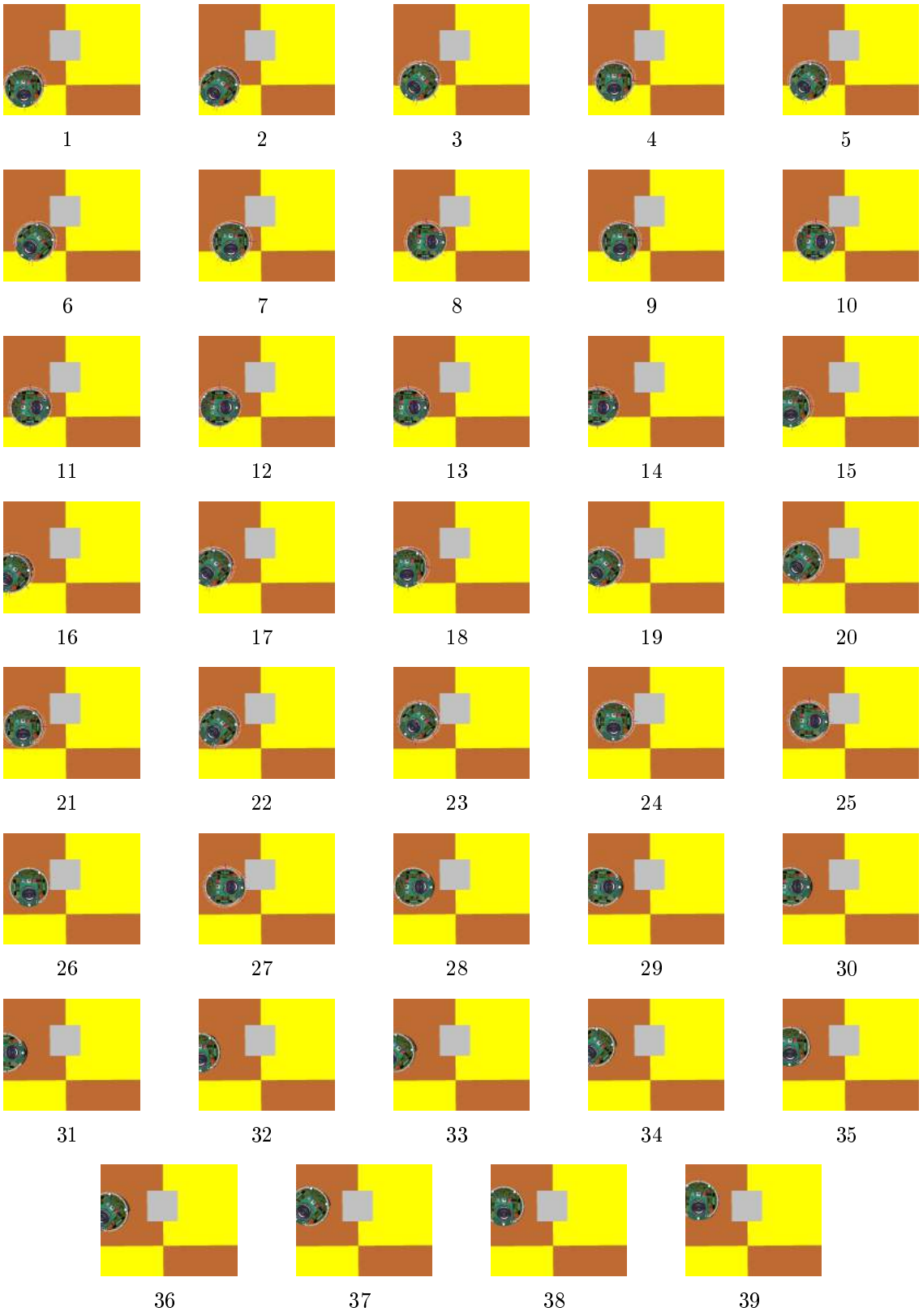
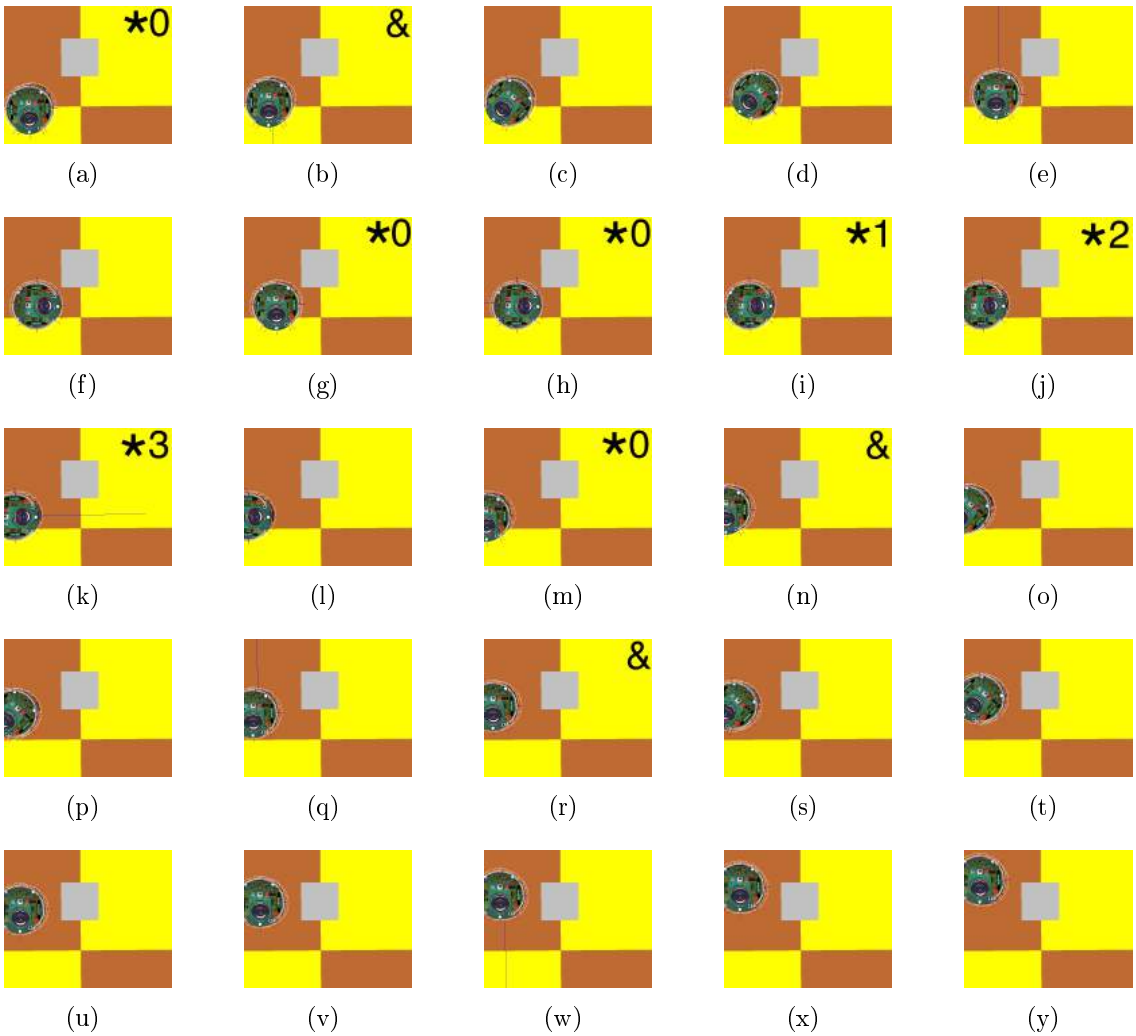


Figura 5.17: Exemplo de percurso onde ocorre o ziguezague quando aplicadas as regras de controle de formação para um único robô.

Células no tempo t											Estado de a0 no tempo t+1	
Caso	a0	a1	a2	a3	a4	a5	a6	a7	a8	Θ_t	a0	Θ_{t+1}
$hor_r - x_r = 0$	Robô	f	f	f	f	f	f	f	f	0°	f	0°
$hor_r - x_r = 0$	f	f	f	f	f	Robô	f	f	f	0°	Robô	0°
$hor_r - x_r > 0$	Robô	f	f	f	f	f	f	f	f	0°	f	0°
$hor_r - x_r > 0$	f	f	f	f	f	Robô	f	f	f	0°	Robô_A	0°
$hor_r - x_r < 0$	Robô	f	f	f	f	f	f	f	f	0°	f	0°
$hor_r - x_r < 0$	f	f	f	f	f	Robô	f	f	f	0°	Robô_A	0°
$hor_r - x_r > 0$	Robô_A	f	f	f	f	f	f	f	f	0°	f	0°
$hor_r - x_r > 0$	f	f	f	f	f	f	Robô_A	f	f	0°	Robô	0°
$hor_r - x_r < 0$	Robô_A	f	f	f	f	f	f	f	f	0°	f	0°
$hor_r - x_r < 0$	f	f	f	f	Robô_A	f	f	f	f	0°	Robô	0°

Figura 5.18: Regras de controle de formação definidas na terceira modificação do modelo.



(a) (b) (c) (d) (e)
(f) (g) (h) (i) (j)
(k) (l) (m) (n) (o)
(p) (q) (r) (s) (t)
(u) (v) (w) (x) (y)

Figura 5.19: Exemplo de percurso após a terceira modificação visando diminuir o ziguezague que ocorre durante o controle de formação com um único robô.

5.4 Experimentos com o robô e-puck real

Após realizarmos as simulações no software Webots avaliando as modificações descritas na seção anterior, a etapa seguinte do trabalho foi a implementação do modelo original e das três variações do novo modelo em um equipamento robótico real. Nesses experimentos, utilizamos a mesma arquitetura das simulações, ou seja, um robô e-puck [École Polytechnique Fédérale de Lausanne EPFL 2013] em sua especificação original.

Com o código já implementado no simulador, a transferência do programa para o equipamento é relativamente simples. Para isso, utilizamos um recurso do e-puck e do Webots que é carregar o programa do simulador no processador a partir de uma comunicação *Bluetooth*. O simulador contém a opção de gerar diretamente um código para ser embarcado. Este processo é chamado *cross-compilation*. Assim, utiliza-se o compilador específico do robô que irá gerar o código que será embarcado. Tendo este código, utiliza-se a opção de *cross-compilation* do Webots para embarcar o código diretamente por *Bluetooth*.

Foram realizados 4 experimentos com o robô e-puck, cada um envolvendo uma versão do modelo: original, conforme o modelo descrito e simulado na Seção 5.2; robo_rotacionado, adicionando a primeira modificação descrita na Seção 5.3.1; robo_rotacionado_i, adicionando a segunda modificação descrita na Seção 5.3.2; e robo_alinhado, adicionando as modificações descritas na seção 5.3.2 e 5.3.3. Os quatro experimentos foram filmados e os filmes encontram-se na página http://www.giordanobsf.com/files/dissertacao/videos-e-puck.zip. As seções seguintes descrevem os quatro experimentos.

5.4.1 Implementação do Modelo Original adaptado para 1 Robô

A primeira implementação foi do modelo original adaptado para ambientes contendo somente 1 robô. Para este caso, vimos em simulação que ocorreu o chamado problema do *deadlock* da quina de obstáculos. Este problema também ocorreu quando o programa foi embarcado em um robô real, como mostra a Figura 5.20.

A Figura 5.20.a mostra o robô no início do seu movimento. Ele então se movimenta dois passo para a frente, como mostrado nas Figuras 5.20.b e 5.20.c, onde ele identifica o obstáculo à frente. Na Figura 5.20.d o robô utiliza uma regra de desvio de obstáculos e faz uma rotação de 90°. À partir da Figura 5.20.e o robô começa a se locomover com os sensores identificando o obstáculo em sua lateral. O movimento continua até a Figura 5.20.h, quando o robô não mais sente um obstáculo em sua lateral, retornando ao ângulo 0° para tentar aplicar uma regra de controle de formação. Contudo, quando retorna ao ângulo 0° (Figura 5.20.i), o robô sente um obstáculo à frente, tendo que refazer uma rotação. As Figuras 5.20.j, 5.20.k, 5.20.l, 5.20.m e 5.20.n mostram o robô rotacionando de 90° para 0° e vice-versa indefinidamente. Assim, o problema que denominamos *deadlock* de quina de obstáculo é também observado nos experimentos reais.

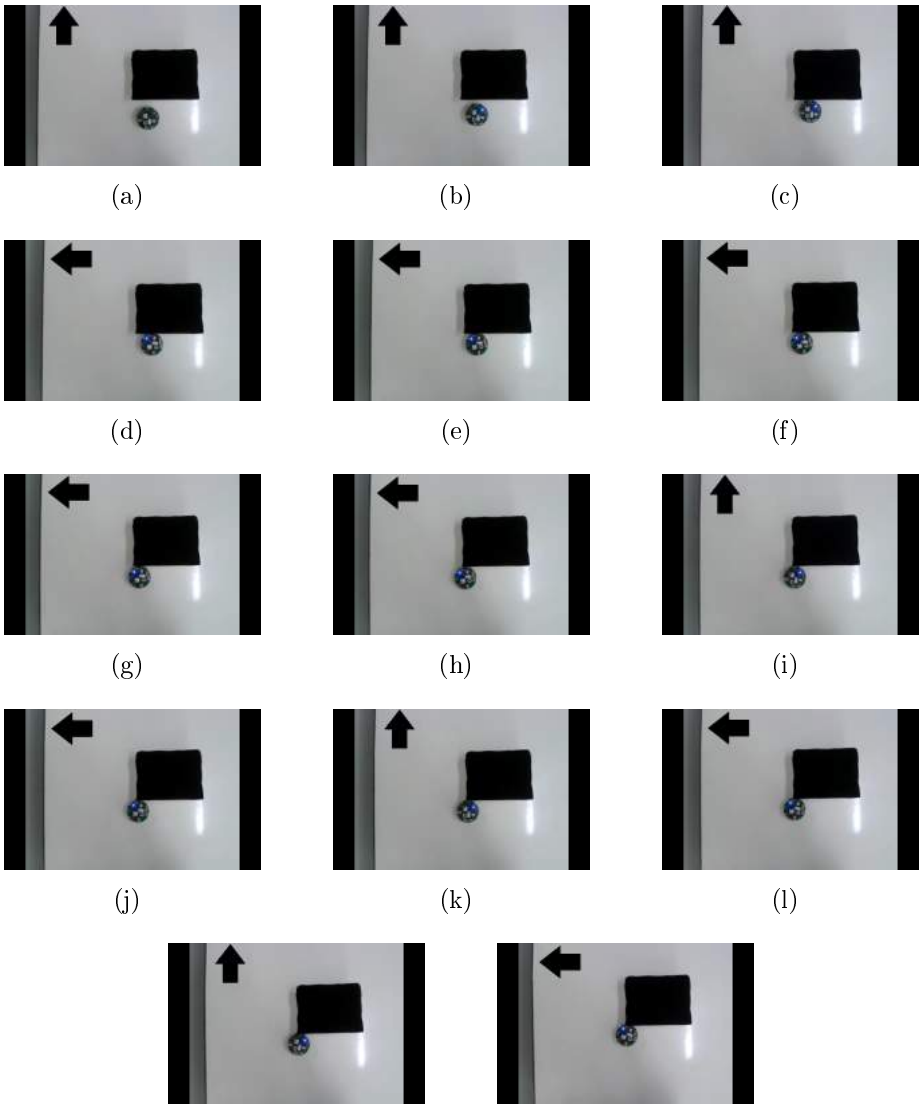


Figura 5.20: Exemplo de execução com um único robô real do modelo original de Ioannidis e colegas.

5.4.2 Implementação do Novo Modelo com o estado Robô_ Rotacionado

A Figura 5.21 apresenta o experimento com o robô se movimentado de acordo com o modelo modificado com a alteração descrita na Seção 5.3.1, que inclui um estado Robô_Rotacionado para evitar o problema do *deadlock*.

A Figura 5.3.1.a apresenta o início do robô em movimento. As figuras 5.3.1.b, 5.3.1.c e 5.3.1.d mostram o robô se movimentando três passos em sua coluna original. Na Figura 5.3.1.e o robô rotaciona para 90° com o objetivo de desviar do obstáculo. As figuras 5.3.1.f, 5.3.1.g, 5.3.1.h e 5.3.1.i mostram o robô se movimentando com o robô identificando o obstáculo em sua lateral. Quando os sensores laterais deixam de identificar o obstáculo, o robô retorna para o ângulo 0° para tentar aplicar alguma regra de controle de formação,

porém, o robô identifica em 5.3.1.j o obstáculo à frente, tendo que rotacionar novamente para 90° em 5.3.1.k, porém agora o estado da célula central é igual a Robô_Rotacionado. Assim, na Figura 5.3.1.l o robô dará um passo para frente com o objetivo de escapar da quina do obstáculo, retornando ao ângulo 0° na Figura 5.3.1.m. Novamente este processo é realizado nas figuras 5.3.1.n, 5.3.1.o e 5.3.1.p, a primeira o robô rotaciona 90°, a segunda ele dará um passo para escapar da quina e a terceira mostra o retorno à sua orientação original. Ao final deste novo passo, o robô não identifica o obstáculo à frente, aplicando uma regra de controle de formação para retornar à sua coluna original. Na Figura 5.3.1.q, o robô rotaciona para $-45°$ para ser possível se movimentar na diagonal. A Figura 5.3.1.r mostra o robô dando um passo na diagonal e a Figura 5.3.1.s mostra o robô retornando à sua orientação original. A Figura 5.3.1.t mostra o robô rotacionando mais uma vez para escapar da quina do obstáculo. Na Figura 5.3.1.u o robô encontra um obstáculo em seus sensores laterais, dando um passo para escapar do obstáculo. Contudo, na Figura 5.3.1.v, novamente o robô não mais encontra obstáculo em sua lateral, retornando à sua orientação original e mudando o estado da célula central para Robô_Rotacionado. Por fim, as figuras 5.3.1.w, 5.3.1.x e 5.3.1.y mostram o robô realizando mais um passo para escapar da quina do obstáculo.

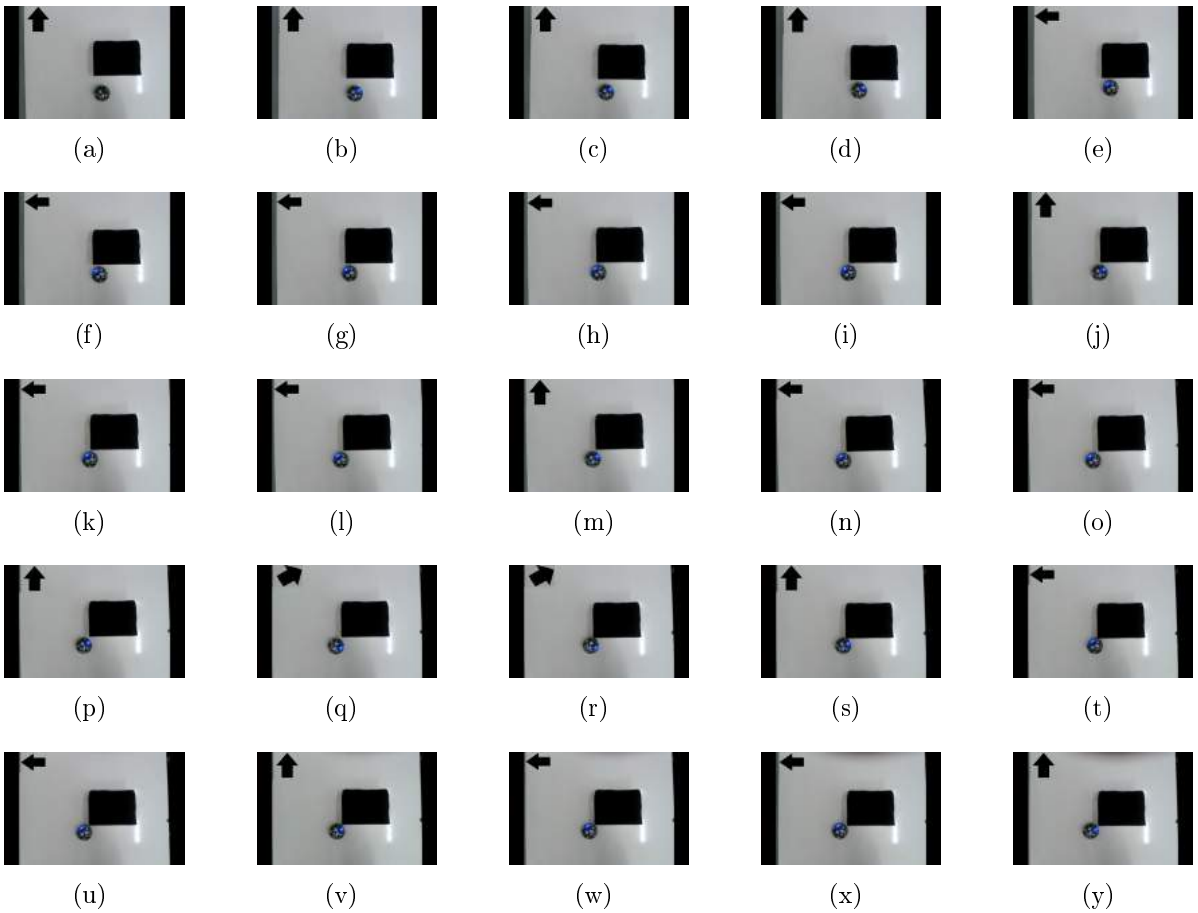


Figura 5.21: Exemplo de execução com um único robô real do modelo com a primeira modificação para evitar o problema do *deadlock*.

5.4.3 Implementação do Novo Modelo com os 4 estados Robô_Rotacionado_i

A segunda modificação do modelo aumentava a quantidade de estados Robô_Rotacionado para uma quantidade i. Nos experimentos com robôs reais do tipo e-puck, foi utilizado o valor i igual a 4, devido ao raio do robô que é um pouco menos que 4 centímetros e o tamanho da célula que é igual a 1 centímetro. Este exemplo o robô deveria dar os quatro passos para escapar da quina do obstáculo, tendo assim uma quantidade menor de giros para escapar, em relação à primeira modificação. A Figura 5.22 mostra um exemplo de percurso para esta modificação.

Como os exemplos anteriores, as primeira figuras mostram o robô se movimentando em sua orientação original, se aproximando assim do obstáculo (figuras 5.22.1, 5.22.2, 5.22.3 e 5.22.4). As figuras 5.22.5, 5.22.6, 5.22.7, 5.22.8 e 5.22.9 mostram o robô desviando do obstáculo. Na Figura 5.22.10, o robô não mais identifica um obstáculo em sua lateral, retornando à sua orientação original, agora com o estado Robô_Rotacionado_0. Agora não existem obstáculos em sua vizinhança, assim ele pode aplicar uma regra de controle de formação. As figuras 5.22.11, 5.22.12 e 5.22.13 mostram o robô rotacionando para $-45°$, se movimentando um passo na diagonal e retornando ao ângulo $0°$. As figuras 5.22.14, 5.22.15 e 5.22.16 mostram o robô se movimentando na lateral do obstáculo novamente até encontrar a quina do obstáculo. O robô volta à sua orientação original em 5.22.17, porém agora com o estado de sua célula central igual à Robô_Rotacionado_0. Em 5.22.18, o robô rotaciona novamente para $90°$ para poder iniciar os 4 passos que serão executados pela nova modificação da regra de desvio, estes passos são mostrados nas figuras 5.22.19, 5.22.20, 5.22.21 e 5.22.22. Como o robô desviou do obstáculo, ele retorna à sua posição original em 5.22.23, para poder começar a utilizar as regras de controle de formação para retornar à sua coluna original. Dois passos da movimentação em diagonal são mostrados nas Figuras 5.22.24, 5.22.25, 5.22.26, 5.22.27, 5.22.28 e 5.22.29. A partir daí, o robô iria novamente aplicar os 4 passos para escapar da quina do obstáculo, resultado em um comportamento de ziguezague.

5.4.4 Implementação do Novo Modelo com os 4 estados Robô_Rotacionado_i e o estado Robô_Alinhado

O último experimento realizado com o robô e-puck foi o da terceira modificação do modelo, que acrescenta um novo estado Robô_Alinhado para auxiliar o controle de formação e diminuir a ocorrência de ziguezagues. Nesta modificação, quando aplicadas as regras de controle de formação, o robô se movimenta por um passo em sua orientação original antes de se movimentar na diagonal para retornar à sua coluna inicial. Um exemplo contendo a implementação desta modificação pode ser visto na Figura 5.23.

Figura 5.22: Exemplo de execução com um único robô real do modelo com a segunda modificação onde quatro passos são dados para escapar da quina do obstáculo.

Neste exemplo, o robô inicia o seu movimento na Figura 5.23.1, se movimentando por 3 passos em direção do obstáculo (figuras 5.23.2, 5.23.3 e 5.23.4). Identificando um obstáculo à sua frente, o robô rotaciona para 90° para desviar do obstáculo em 5.23.5. Ele então inicia os movimentos de desvio do obstáculo com o obstáculo sendo identificado pelo sensor lateral (figuras 5.23.6, 5.23.7 e 5.23.8). Em seguida, o robô retorna à sua orientação original por não mais identificar o obstáculo na lateral, porém agora o estado de sua célula central é igual à Robô_Rotacionado_0. Assim, em 5.23.10 o robô retorna à 90° para iniciar os 4 passos para escapar da quina do obstáculo. Estes passos são mostrados nas figuras 5.23.11, 5.23.12, 5.23.13 e 5.23.14. Em 5.23.15, o robô retorna à sua orientação original, podendo aplicar as regras de controle de formação, pois não existem obstáculos em sua vizinhança. A nova regra de controle de formação define que o robô agora dê um passo à frente antes de se movimentar em diagonal, e isto é o que de fato ocorre na Figura 5.23.16. O novo estado da célula central do robô é igual à Robô_Alinhado, portanto ele pode agora se movimentar na diagonal. O movimento em diagonal pode ser

visto nas figuras 5.23.17, 5.23.18 e 5.23.19. Em 5.23.20, o robô identifica um obstáculo à frente, rotacionando para 90°, contudo em 5.23.21 o robô não identifica obstáculo em sua lateral, retornando ao ângulo 0°, com o estado Robô_Rotacionado_0. A rotação para 90° é novamente realizada em 5.23.22, e agora o robô poderá aplicar os 4 passos para escapar da quina do obstáculo. Em 5.23.27, o robô não encontra obstáculos em sua vizinhança e novamente poderá aplicar as regras de controle de formação, dando um passo à frente (Figura 5.23.28) e se movimentando em diagonal (figuras 5.23.29, 5.23.30 e 5.23.31). À partir deste momento, o robô deveria encontrar um obstáculo em sua lateral, aplicando assim as regras de desvio de obstáculos e diminuindo o ziguezague no desvio do obstáculo, pois não seria mais necessário aplicar mais 4 passos para escapar do obstáculo. Contudo, nos experimentos reais, devido aos erros acumulados da odometria, este comportamento não foi visto. Se continuarmos o processo após a Figura 5.23.31, o robô dará novamente 4 passos para desviar do obstáculo, mantendo o comportamento de ziguezague.

5.4.5 Comentários sobre os experimentos com o robô e-puck

Após as implementações, foi mostrado que as modificações eram mesmo necessárias para a devida aplicação do modelo em um ambiente com um robô e-puck. A primeira modificação possibilitou que o robô escapasse da situação de *deadlock* na quina do obstáculo, porém sendo necessário várias rotações para que de fato o robô atravessasse completamente o obstáculo. A segunda modificação diminuiu a quantidade de rotações, pois o robô identifica que está em uma quina de obstáculo caminhando assim os 4 passos necessários para escapar desta quina. Porém, enquanto tentava retornar à sua coluna original, às vezes o robô necessitava aplicar os 4 passos para afastar da quina novamente, ocorrendo assim o problema do ziguezague no desvio de obstáculos. A última modificação visava a diminuição deste comportamento de ziguezague, pois os passos que ele aplicaria em sua orientação original quando utilizava as regras de controle de formação faria com que ele encontrasse o obstáculo novamente com seu sensor lateral, podendo aplicar a regra de desvio de obstáculos para se movimentar com o obstáculo estando em sua lateral. Porém nos experimentos reais, devido ao erro da odometria, este comportamento não foi de fato observado.

Assim, é necessário que para os próximos experimentos a odometria e os sensores sejam melhor calibrados. Ocorreram casos durante os experimentos em que a rotação não é aplicada pela quantidade desejada, este erro se acumula e após alguns passos, o robô tem a informação que está em uma posição muito distante da que ele de fato está. Os sensores também devem ser melhor calibrados para que o robô só identifique obstáculos que estão de fato a uma célula de distância.

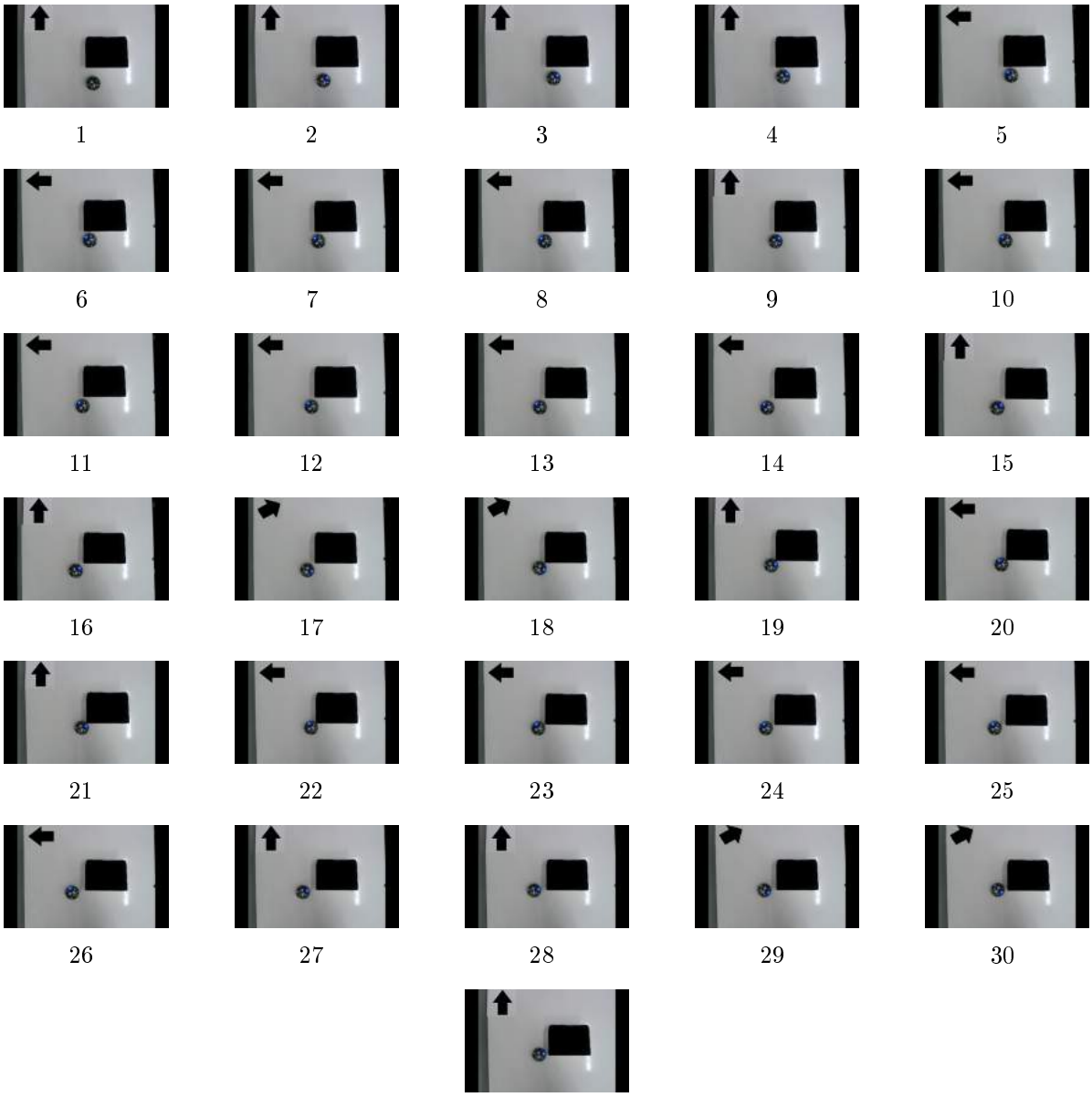


Figura 5.23: Exemplo de execução com um único robô real do modelo com a terceira modificação para evitar comportamento de ziguezague.

5.5 Análise do Modelo Cooperativo Original

Após a análise e implementação do novo modelo em cenários com um único robô no ambiente, descrita na Seção 5.3, o foco da dissertação passou a ser a implementação do modelo envolvendo cenários com um time de robôs e com a formação em linha reta. Dessa forma, a comunicação entre os robôs foi implementada no ambiente de simulação Webots, para que fosse possível executar o modelo original proposto em [Ioannidis et al. 2008] com três robôs e o controle de formação.

5.5.1 Implementação da Comunicação entre os Robôs

A comunicação entre os robôs foi feita através do protocolo *Bluetooth*, que é implementado por padrão nas arquiteturas de robôs e-pucks [École Polytechnique Fédérale de Lausanne EPFL 2013]. Esta comunicação é implementada no ambiente de simulação Webots [Cyberbotics 2013] através de duas classes *Emitter* e *Receiver*. Portanto foi possível simular o comportamento cooperativo em simulação pelo Webots. A primeira necessidade foi implementar o protocolo de comunicação entre os robôs. Três mensagens foram necessárias para a comunicação entre os robôs. A primeira, era enviada pelo robô escravo, informando ao mestre sua localização no ambiente. A segunda, era a resposta da primeira mensagem, onde o robô mestre informa ao escravo qual deve ser o seu comportamento no próximo passo de tempo, ou seja, informa o novo estado da célula central, além do ângulo θ de rotação. A terceira mensagem foi necessária durante as implementações, era enviada pelo mestre sempre que uma troca de mestre ocorria, desta forma o outro robô escravo que não estava envolvido na troca consegue saber qual o novo robô mestre.

A primeira mensagem foi implementada como mostrada em [Ioannidis et al. 2011b], onde foi necessário uma mensagem contendo três bytes informando o estado da célula, que serve também como identificador da mensagem, sua posição horizontal no reticulado e por último sua posição vertical. Estas três informações são necessárias para que o robô mestre possa decidir qual a regra de transição o robô escravo deve utilizar. A segunda mensagem não foi definida nos artigos de Ioannidis e colaboradores encontrados. Portanto, a resposta enviada pelo robô mestre para os seus robôs escravos foi definida em quatro bytes. O primeiro é um identificador a quem esta resposta está endereçada, o segundo contém o novo estado da célula central do robô e os dois últimos contém o ângulo de rotação - em graus - que o robô deve estar ao final deste passo de tempo.

A terceira mensagem também não foi definida nos trabalhos, mas durante os experimentos ela se fez necessária. Assumindo R_2 o robô mestre, R_1 o da esquerda e R_3 o robô à direita na formação, o problema ocorria no seguinte cenário: R_3 enviava sua posição e aguardava a resposta do mestre, R_1 enviava sua posição e aguardava a resposta do mestre, R_2 recebia a mensagem de R_3 e decidia por uma troca de posições. Neste momento, R_2 é um escravo com uma mensagem de R_1 pendente de tratamento. R_3 que agora é o novo mestre, na época era um escravo e também não tratou a mensagem de R_1. Assim, R_1 fica indefinidamente à espera da resposta para sua próxima ação. A terceira mensagem consiste em dois bytes, sendo o primeiro o identificador desta mensagem e o segundo o id do novo mestre. Quando ocorre a troca de posições entre dois robôs, o antigo robô mestre manda esta mensagem informando o id do novo mestre, assim, o robô que está à espera de uma resposta para seu próximo movimento pode enviar uma nova mensagem contendo sua posição com destino este novo mestre.

Como cada robô executa sua implementação independente (mesmo em simulação),

quando a formação não está perfeita, pode acontecer tempo de movimentação ser distinto entre os robôs. Não existe de fato um "sincronizador" de movimentos, assim o conceito de passo de tempo vale para o escopo de cada robô. Por exemplo, um robô deseja movimentar para frente e o outro deseja andar a 45°, em um passo de tempo o primeiro realiza o seu movimento, enquanto o segundo necessita de três (um para rotacionar para 45°, outro para dar um passo e um terceiro para rotacionar novamente para 0°). Porém, quando a formação está perfeita, todos os robôs dão somente um passo de tempo cada, então o único atraso é o relativo à troca de mensagens - imperceptível em simulação.

5.5.2 Implementação do Modelo Cooperativo Original

Um exemplo de percurso para o time de robôs com formação em linha, de acordo com o modelo proposto em [Ioannidis et al. 2008], pode ser visto de forma esquematizada na Figura 5.24. Na Figura 5.24.a temos o grupo de robôs direcionados para o norte e alinhados em formação, sendo que o robô central é o mestre do time. Nesse exemplo, o valor de *dist* é igual a 3; portanto, cada robô está distante do outro em três células. Denominaremos o robô a esquerda como R_1, o central como R_2 e o da direita como R_3. Na figura eles são representados pelas cores azul, amarelo e vermelho respectivamente. Logo neste primeiro passo, o robô R_1 identifica um obstáculo em suas células a_1 e a_8, e aplica uma regra de desvio de obstáculos sendo necessário uma rotação. A Figura 5.24.b, mostra que R_1 fez sua rotação para $-90°$ e que R_2 e R_3 se comunicaram e, como não se aproximaram, deram um passo em direção a meta. Nas figuras 5.24.c e 5.24.d, R_1 aplica as regras de desvio para dar mais um passo para sair do obstáculo que estava à frente, enquanto isso, R_2 e R_3 aplicam a regra de controle de formação e dão mais um passo em direção à meta. Na Figura 5.24.e, o robô R_1 completou o desvio de obstáculo, e volta a sua orientação original. Nesse momento, R_1 se comunica com R_2 e como os dois se aproximaram, devem trocar de posição na formação, sendo que R_1 se torna o mestre da formação e R_2 se torna um escravo. Devido à troca de posições, R_2 deve se movimentar até a antiga coluna de R_1 e vice-versa. Neste passo, R_3 mantém sua posição, pois se comunica com o novo robô mestre e verifica que está à frente. A Figura 5.24.g mostra que R_1 chegou à sua nova coluna e R_2 está agora mais próximo de sua nova coluna, R_3 ainda deve esperar mais um passo. Já na Figura 5.24.h, R_1 se movimenta um passo em direção à meta enquanto R_3 espera por mais um passo de tempo. Enquanto isso, R_2 continua sua movimentação em direção à sua coluna até que na Figura 5.24.i ele finalmente chega em sua nova coluna, e R_1 se aproxima mais uma célula da meta enquanto R_3, ainda espera. As Figuras 5.24.j, 5.24.k, 5.24.l e 5.24.m mostram os robôs R_1 e R_3 se movimentando em direção à meta enquanto R_2 continua à espera dos outros robôs para voltar a se movimentar. Por fim, nas Figuras 5.24.n contém os robôs terminando suas movimentações cooperativamente até a meta.

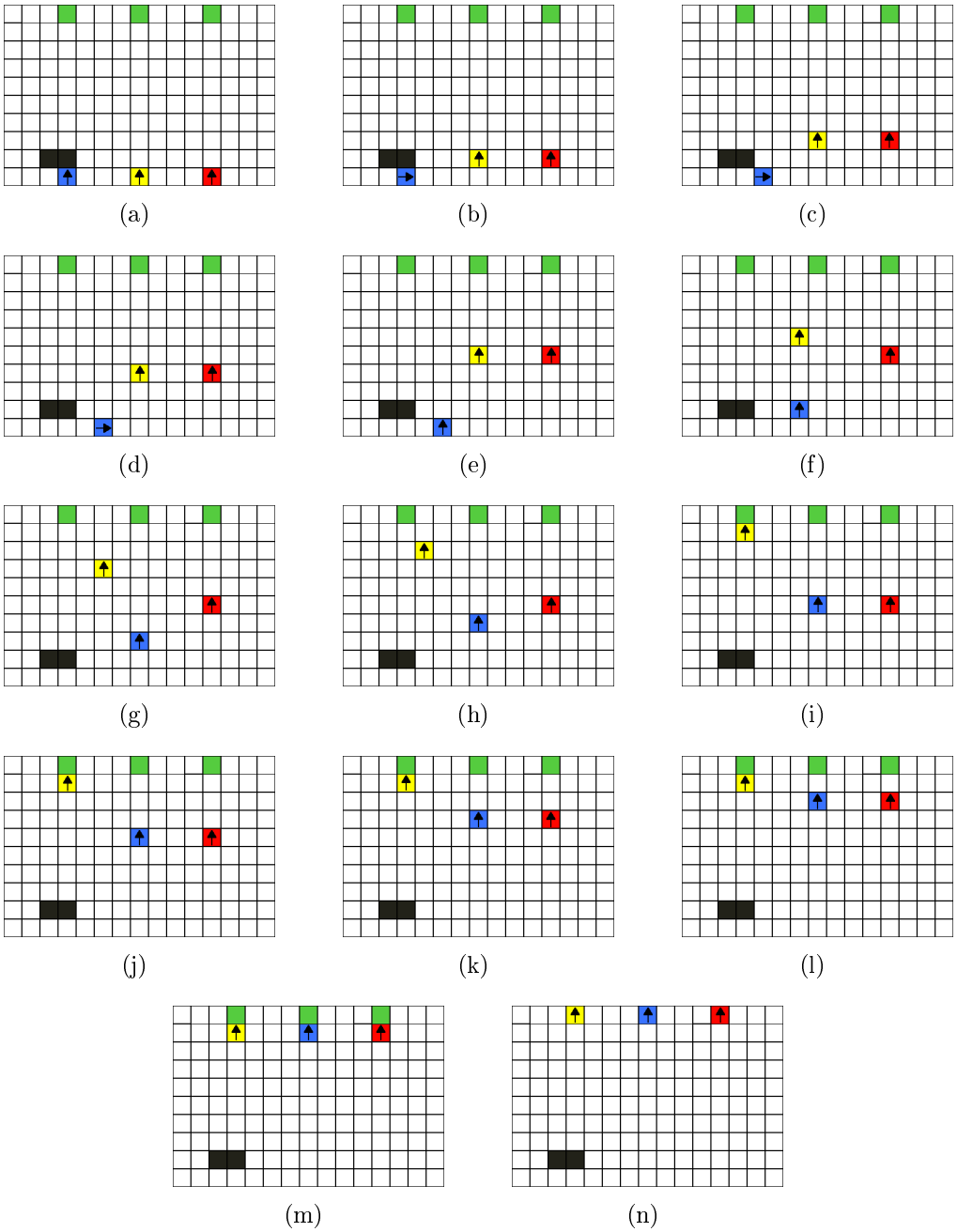


Figura 5.24: Exemplo de percurso em ambiente com um time de robôs. Células azul, amarela e vermelha representam os robôs, sendo a seta a indicação da orientação naquele passo de tempo, a célula verde a meta, e as células pretas obstáculos.

Através da execução de diversas simulações utilizando o método descrito em [Ioannidis et al. 2008] no ambiente de simulação Webots, identificamos um problema que pode ocorrer quando se utiliza o modelo original. Apesar de não acontecer em uma frequência tão comum como o problema do deadlock da quina do obstáculo, é um problema crítico que também impossibilita o time de robôs de alcançar a meta de forma cooperativa. Chamamos esse problema de "pareamento de robôs". A Figura 5.25 mostra um segundo exemplo de forma esquemática, no qual o problema do pareamento ocorre. Nele, o robô mestre da formação encontra um obstáculo à frente, fazendo com que ele se desvie em

direção ao robô escravo à esquerda. As figuras 5.25.a, 5.25.b, 5.25.c e 5.25.d mostram o robô mestre desviando do obstáculo, enquanto os robôs escravos esperam o seu próximo comando. Após o desvio do obstáculo, os dois robôs que se aproximaram se comunicam para aplicarem as regras de controle de formação, assim na Figura 5.25.f ambos os robôs iniciam a troca de posição. Neste caso ocorre o problema, pois agora ambos os robôs estão lado a lado no mapa, e os sensores de ambos tratam um ao outro como um obstáculo no ambiente. Assim, ambos irão utilizar alguma regra de desvio de obstáculo independentemente, se movimentando para frente, como mostrado na Figura 5.25.g, e no próximo passo de tempo. Dessa forma, ambos continuarão com o obstáculo ao lado indefinidamente e não conseguirão realizar a troca de posições.

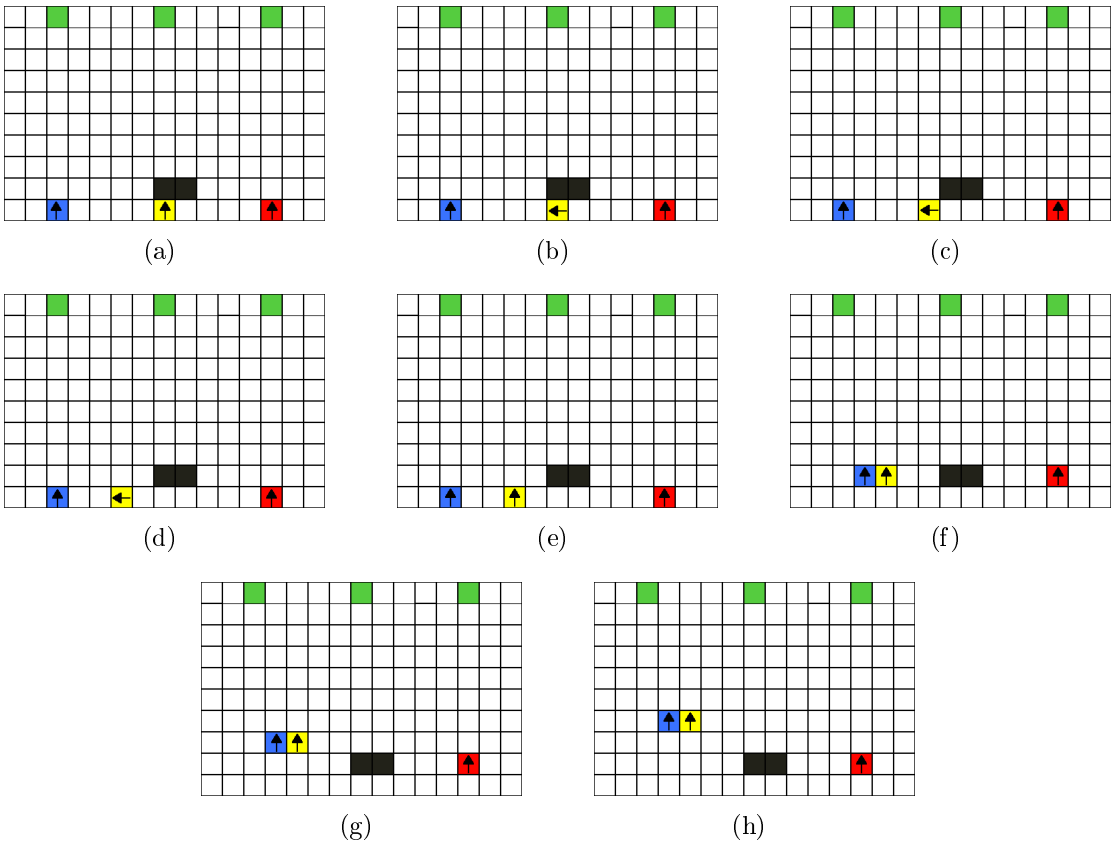
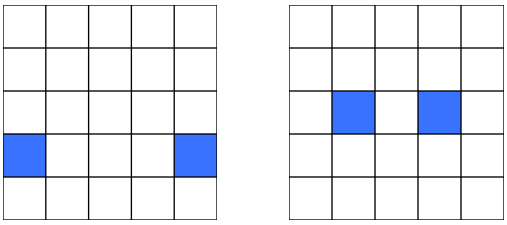


Figura 5.25: Exemplo de problema que ocorre no método com um time de robôs. Células azul, amarela e vermelha representam os robôs, sendo a seta a indicação da orientação naquele passo de tempo, a célula verde a meta, e as células pretas obstáculos.

Um problema ainda mais sério pode também acontecer durante a troca de colunas, caso dois robôs decidam ir para a mesma célula no mesmo passo de tempo. Neste caso, ocorre a colisão entre os robôs, o que é ainda mais danoso para o modelo. A Figura 5.26 ilustra essa situação. Nesse cenário, imagine que os dois robôs estão progredindo na diagonal, para realizar a troca das posições. Vemos da Figura 5.26.a para a 5.26.b que os dois robôs caminham na diagonal para se deslocarem na direção de suas novas colunas de referência. Na Figura 5.26.b a leitura dos sensores de ambos não identificará qualquer

obstáculo e no próximo passo os dois tentarão dar um passo progredindo na diagonal e colidirão na mesma célula. Embora esse problema de colisão seja ainda mais problemático, ele não é tão frequente quanto o problema de pareamento descrito anteriormente. Ambos os problemas são mais prováveis de ocorrer no modelo original devido a essa estratégia de mudança de colunas de referência, inerente ao modelo de Ioannidis e colaboradores.



(a) (b)

Figura 5.26: Exemplo de caso onde ocorre colisão entre os robôs durante a troca de colunas.

5.5.3 Simulação do Modelo Original "Adaptado"

Retornando aos artigos de Ioannidis e colaboradores, o problema do pareamento dos robôs, ou mesmo o da colisão, não foi identificado nos experimentos descritos nos artigos. Porém, acreditamos que o motivo de não ter sido identificado foi uma diferença na forma em que o método foi explicado e como ele de fato foi implementado, pois o comportamento dos robôs nas figuras dos artigos difere do algoritmo da forma como ele foi explicado. Entretanto, não foi possível concluir exatamente como se deu a implementação a partir da observação exclusiva das figuras. Por isso, optamos por implementar o modelo seguindo a descrição apresentada pelos autores. Adicionalmente, para que fosse possível a aplicação deste método em simulação com robôs e-puck e com os robôs reais, alguma modificação deve ser utilizada para superar o problema do deadlock na quina do obstáculo, que ocorre sempre que algum robô do time precisa desviar de um obstáculo. Essa limitação na descrição do modelo original, impedia que fossem realizadas simulações em cenários com obstáculos, por mais simples que fossem. Assim, optamos por implementar o modelo descrito no artigo acrescentando a primeira modificação descrita na Seção 5.3.1, que inclui o estado Robo_Rotacionado. O resultado de uma simulação com essa implementação pode ser visto na Figura 5.27.

Denominaremos os robôs da esquerda para a direita no cenário inicial de R_1, R_2 e R_3, sendo que no início R_2 é o mestre e R_1 e R_3 os escravos. O início do processo, mostra o robô mestre R_2 com um obstáculo à frente, portanto ele inicia aplicando as regras de desvio de obstáculos. Assim, até a Figura 5.27.i, temos os passos em que o robô mestre está desviando do obstáculo, enquanto os dois escravos esperam. A partir da Figura 5.27.j, o R_2 não identifica o obstáculo na vizinhança e começa assim a troca de posições com o robô à sua esquerda (R_1). Porém, na Figura 5.27.n, o novo robô escravo (R_2) encontra o novo

robô mestre (R_1) em sua lateral. Como só são utilizados os sensores de infra-vermelho para a construção da vizinhança, a identificação de um outro robô é interpretada um obstáculo qualquer. Assim, na Figura 5.27.o, o robô escravo (R_2) inicia a aplicação das regras de desvio de obstáculos, caminhando para frente para desviar de R_1. Ele continua caminhando até a Figura 5.27.v. Especificamente neste exemplo de simulação, o robô mestre R_1 não chegou a identificar R_2 a seu lado, pois estava aplicando as regras de controle de formação e caminhando em diagonal, enquanto o robô R_2 atravessava na vertical. Por isso, nessa simulação, embora as figuras não apresentem o final da simulação, o robô R_1 consegue atravessar na diagonal até chegar em sua nova coluna, enquanto R_2 percorre um pouco mais na vertical e depois passa a se deslocar também na diagonal até sua nova coluna de referência. Assim, os robôs conseguiram efetuar a troca de colunas nessa simulação. Entretanto, caso o robô R_1 também tivesse identificado o robô R_2 quando eles se aproximaram, ele teria mudado para a regra de desvio de obstáculo também e ambos iriam caminhar lado-a-lado indefinidamente.

A Figura 5.28 apresenta um segundo exemplo de simulação que parte de um cenário inicial bastante similar ao da Figura 5.27. Nessa segunda simulação, o problema do pareamento de fato ocorreu, como pode ser visto à partir da Figura 5.27.28.

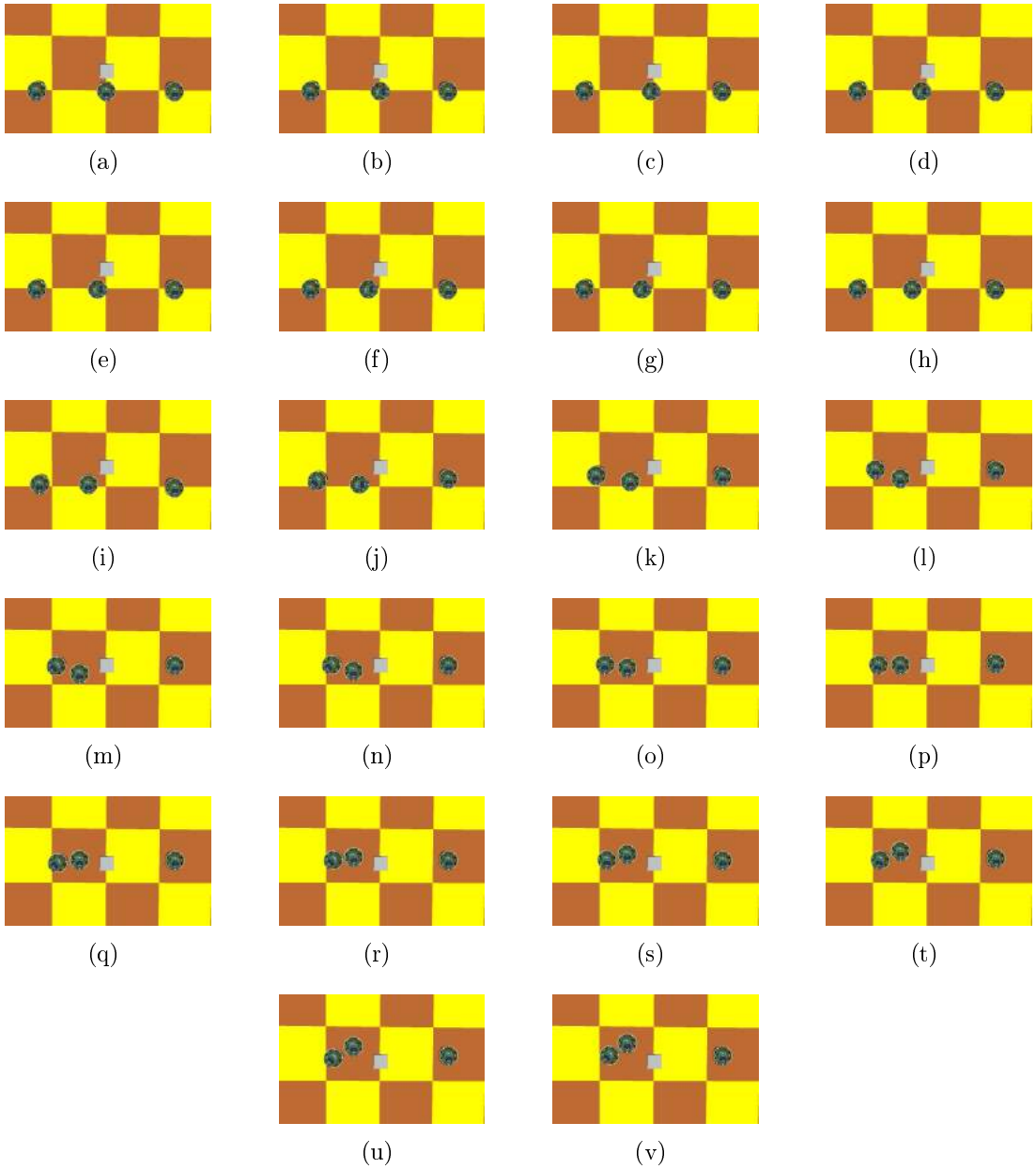


Figura 5.27: Exemplo no simulador Webots da forma descrita por Ioannidis e colaboradores para um exemplo onde ocorre o caso dos robôs com problema para trocarem de posição.

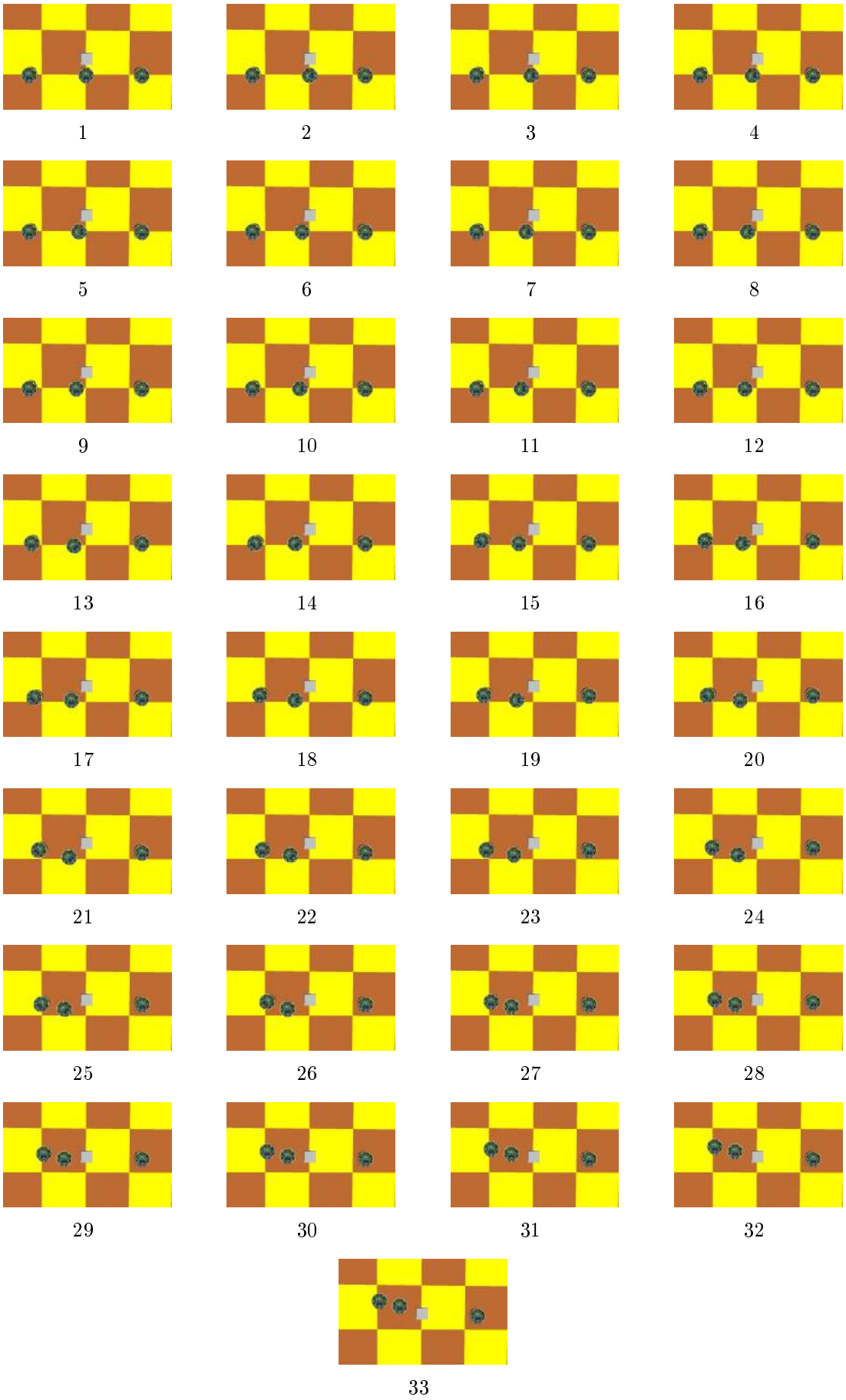


Figura 5.28: Exemplo de percurso onde ocorre o problema do pareamento de robôs.

5.6 Aplicação do Novo Modelo Cooperativo Baseado na Regra de Atualização Local para cenários com o time de robôs

Na seção anterior, foi ilustrado o comportamento de um time de robôs cooperando, de acordo com o modelo original de Ioannidis e colaboradores, no qual ocorre a troca de colunas de referência sempre que o robô mestre identifica que se aproximou de um escravo. Numa etapa subsequente de nossa pesquisa, decidimos implementar o modelo com o time de robôs sem a troca das colunas de referência em casos de aproximação, utilizando o mesmo modelo utilizado nos cenários contendo apenas 1 robô descrito na Seção 5.3.

A comunicação entre os robôs foi implementada de forma similar à descrita no modelo original. Porém, em relação ao controle de formação, como no novo modelo cooperativo não existe a troca das colunas de referência, o pseudocódigo foi modificado e é apresentado na Figura 5.29. Em relação ao pseudocódigo apresentado anteriormente na Figura 5.10, a mudança principal é que não existe mais a necessidade de tratar de forma diferenciada quando os robôs se aproximam para realizar a troca de coluna ou permitir que robôs caminhando em diagonal avancem ainda que exista uma parte do time atrasada. Além disso, o robô mestre só se movimenta quando está em uma linha anterior do que os dois robôs ao mesmo tempo. Ou seja, os robôs que estejam em controle de formação (sem identificar obstáculos na vizinhança), mas estejam a frente de outros membros do time, sempre aguardam até que os demais membros estejam na mesma linha.

A Figura 5.30 ilustra de forma esquemática como funciona o novo modelo utilizando o mesmo cenário inicial apresentado na figura 5.25. Esse modelo não aplica a troca das colunas em caso de aproximação, conforme o pseudocódigo da Figura 5.29, e também incorpora a primeira modificação descritas nas seções 5.3.1, com a inclusão do estado Robo_Rotacionado para evitar o *deadlock* de quina de obstáculo.

É possível perceber pelo exemplo da Figura 5.30, que no novo modelo sem a troca das colunas, o problema dos robôs se deslocarem lado a lado não mais ocorreu, sendo possível aos robôs retornarem à formação. A principal justificativa em [Ioannidis et al. 2011a] para a utilização da troca de colunas de referência em caso de aproximação é para que o robô retorne para a sua nova coluna o mais rápido o possível e assim possa voltar a caminhar coordenadamente com os outros robôs da formação. Se compararmos o novo método sem troca de coluna com o anterior de Ioannidis e colaboradores, verificamos que neste caso específico, o robô não conseguia encontrar a meta no modelo original, sendo possível encontrá-la agora. Quando o exemplo possibilita que não ocorra o problema dos robôs lado-a-lado, o modelo original de fato consegue realizar a troca de posições em uma quantidade menor de passos.

A Figura 5.31 apresenta a navegação dos 3 robôs no novo modelo cooperativo partindo-

```
//m = robo mestre
//e = robo escravo da esquerda
//d = robo escravo da direita
if yd == ye then
      if ym == ye then
            aplica_formacao(m)
            aplica_formacao(e)
            aplica_formacao(d)
      else
            if ym < ye then
                  aplica_formacao(m)
            else
                  aplica_formacao(e)
                  aplica_formacao(d)
            end-if
      end-if
else
      if yd < ye then
            aplica_formacao(d)
            if ym <= yd then
                  aplica_formacao(m)
            end-if
      else
            aplica_formacao(e)
            if ym <= ye then
                  aplica_formacao(m)
            end-if
      end-if
end-if
```

Figura 5.29: Pseudo-código onde não ocorre a troca das colunas de referência quando os robôs se aproximam.

se do mesmo cenário da Figura 5.24, na qual o modelo original, com troca de colunas, conseguiu alcançar a meta. No caso do modelo original (Figura 5.24), em 13 passos os robôs se encontram na meta. No modelo modificado (Figura 5.31), os robôs necessitam de 16 passos. Contudo, devido à troca de posições, o número de passos onde a formação é mantida no modelo original, mesmo que com uma distância horizontal diferente de *dist*, é menor do que no novo modelo, pois no modelo original, os robôs se movimentam independente da formação quando devem realizar a troca, como visto na Figura 5.24.

Posteriormente, o novo modelo cooperativo foi avaliado no ambiente de simulação Webots. O cenário inicial utilizado na simulação apresentada na Figura 5.32 é o mesmo da simulação da Figura 5.27. Nessa simulação, foi empregado o novo modelo cooperativo sem troca de colunas, explicado anteriormente. Entretanto, na simulação da Figura 5.32 foi utilizada a segunda modificação do modelo descrita para o caso de robô único (Seção 5.3.2). Ou seja, foi realizada a inclusão de 4 estados Robo_Rotacionado_i no desvio de obstáculos. Para a redução da quantidade de cenários apresentados na figura, que seria muito extenso nesse exemplo, vários passos estão compreendidos entre dois cenários consecutivos.

Iniciando na Figura 5.32.a, temos o robô mestre com o obstáculo à frente. Em 5.32.b,

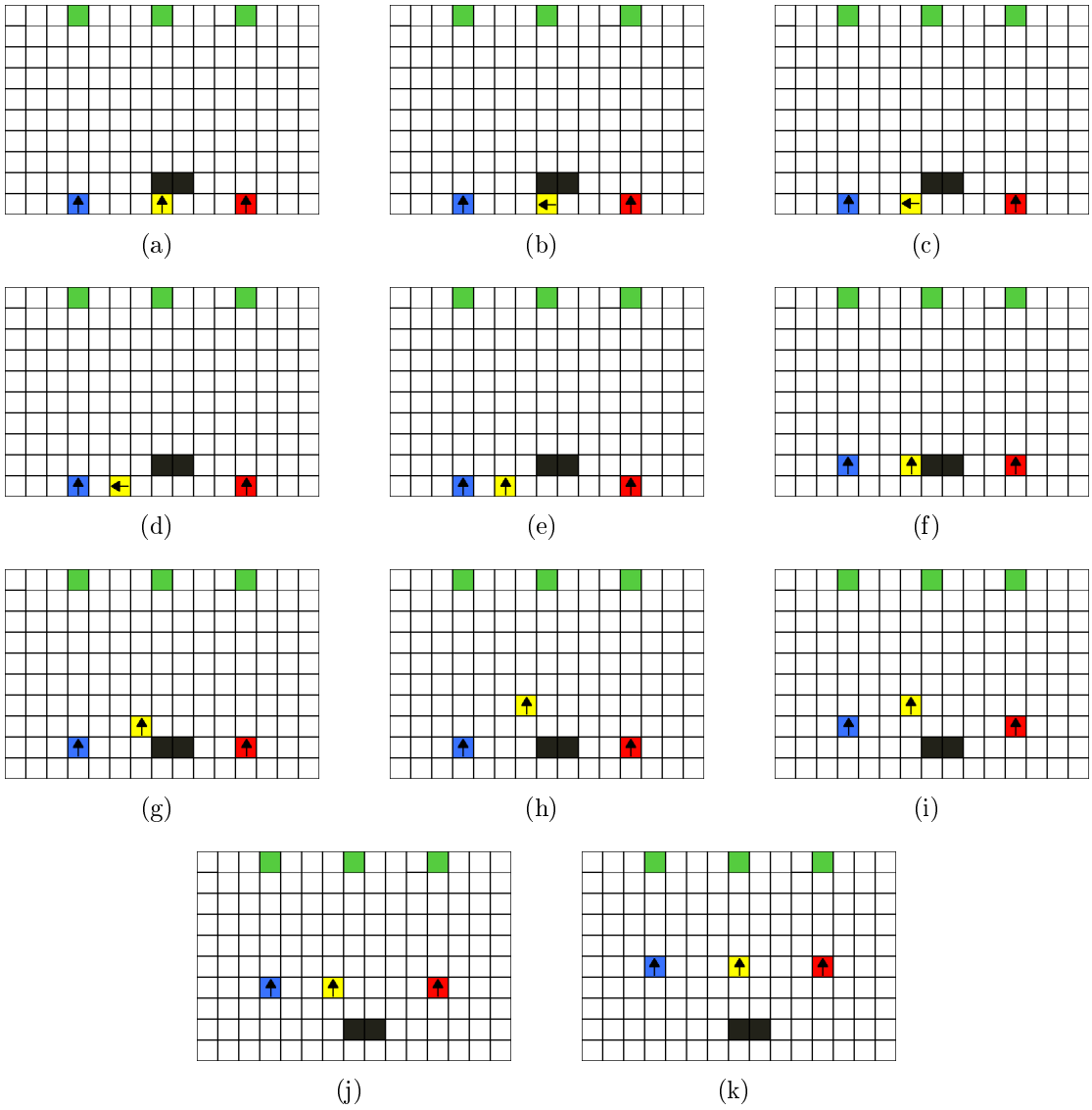


Figura 5.30: Exemplo utilizando o algoritmo sem a troca de posições.

ele faz a rotação para iniciar o movimento de saída do obstáculo. A Figura 5.32.c, mostra o robô após aplicar as regras de desvio de obstáculos modificadas de acordo com a Seção 5.3.2 (ou seja, 4 passos à esquerda para escapar da quina). A Figura 5.32.d contém o robô após realizar um passo na diagonal para retornar à sua coluna de origem. Este passo ocorre efetivamente em três movimentos, rotação para $-45°$, movimentação para frente e por fim a rotação de volta para $0°$. Neste passo, o robô à esquerda também dá um passo à frente. Na Figura 5.32.e o robô volta a identificar um obstáculo em sua vizinhança e aplica novamente as regras de desvio até a Figura 5.32.f. As figuras 5.32.g, 5.32.h, 5.32.i e 5.32.j apresentam o robô aplicando o controle de formação para retornar à sua coluna original, enquanto que os robôs escravos também dão passos em direção à meta. A Figura 5.32.k contém o robô novamente identificando um obstáculo ao lado e iniciando o ziguezague, pois ele terá que novamente aplicar as regras de desvio de obstáculo. As figuras 5.32.l e 5.32.m contém o robô aplicando as regras de controle de formação novamente, após o

desvio de 4 passos a esquerda. Na Figura 5.32.n, o robô identifica o obstáculo em sua lateral e começa a caminhar em frente, até a Figura 5.32.o, quando ele passa a esperar os outros robôs encontrarem a sua linha. Na Figura 5.32.p, os escravos chegam até sua linha e então o grupo pode voltar a andar cooperativamente. Da Figura 5.32.q até a 5.32.v, o robô mestre começa a voltar para a sua coluna aplicando deslocamentos na diagonal, enquanto os robôs escravos fazem deslocamentos em linha reta até que na Figura 5.32.y, o robô mestre encontra sua coluna original. Finalmente, na Figura 5.32.z os três robôs se movimentam em linha com a distância entre eles igual à definida no início do processo. Assim, é possível perceber nesse exemplo que no novo modelo cooperativo os robôs conseguem retornar à formação original utilizando uma estratégia mais simples do que a troca de colunas. Além disso, o comportamento do modelo sem o cruzamento das rotas dos robôs, que é provocado no modelo original pela estratégia de troca de colunas, evita a ocorrência das situações de pareamento e colisão de robôs.

Pelos resultados das simulações, concluímos que a troca de posição na formação não é de fato tão eficiente quanto os autores descrevem. Apesar dos robôs de fato retornarem à suas novas colunas o mais rápido o possível, são necessários muitos passos para que os robôs voltem a se movimentar mantendo a formação. Utilizando as regras desenvolvidas para o modelo com um único robô no ambiente, que não forçam a troca de dois robôs ao se aproximarem, o tempo de retorno de um robô que desvia de um obstáculo à sua coluna é um pouco superior, porém a quantidade de passos onde os robôs se movimentam em linha, é maior, retornando um comportamento mais eficiente do time como um todo. Além disso, o novo modelo evita o problema dos robôs se chocarem durante a troca de posições, fato que ocorreu em nossa implementação.

Para a implementação do modelo cooperativo com robôs reais, é necessário que se implemente a comunicação via *Bluetooth* entre os robôs. Em simulação, esta comunicação é feita através de duas classes (*Emitter* e *Receiver*) fornecidas pelo simulador Webots. Estas classes não são possíveis de serem embarcadas nos robôs reais, sendo necessária que a implementação seja feita pelo desenvolvedor. Esta implementação começou a ser estudada mas não houve tempo hábil para finalizá-la no escopo da dissertação. Assim, os testes do modelo cooperativo com robôs reais deverão ser realizados em trabalhos futuros.

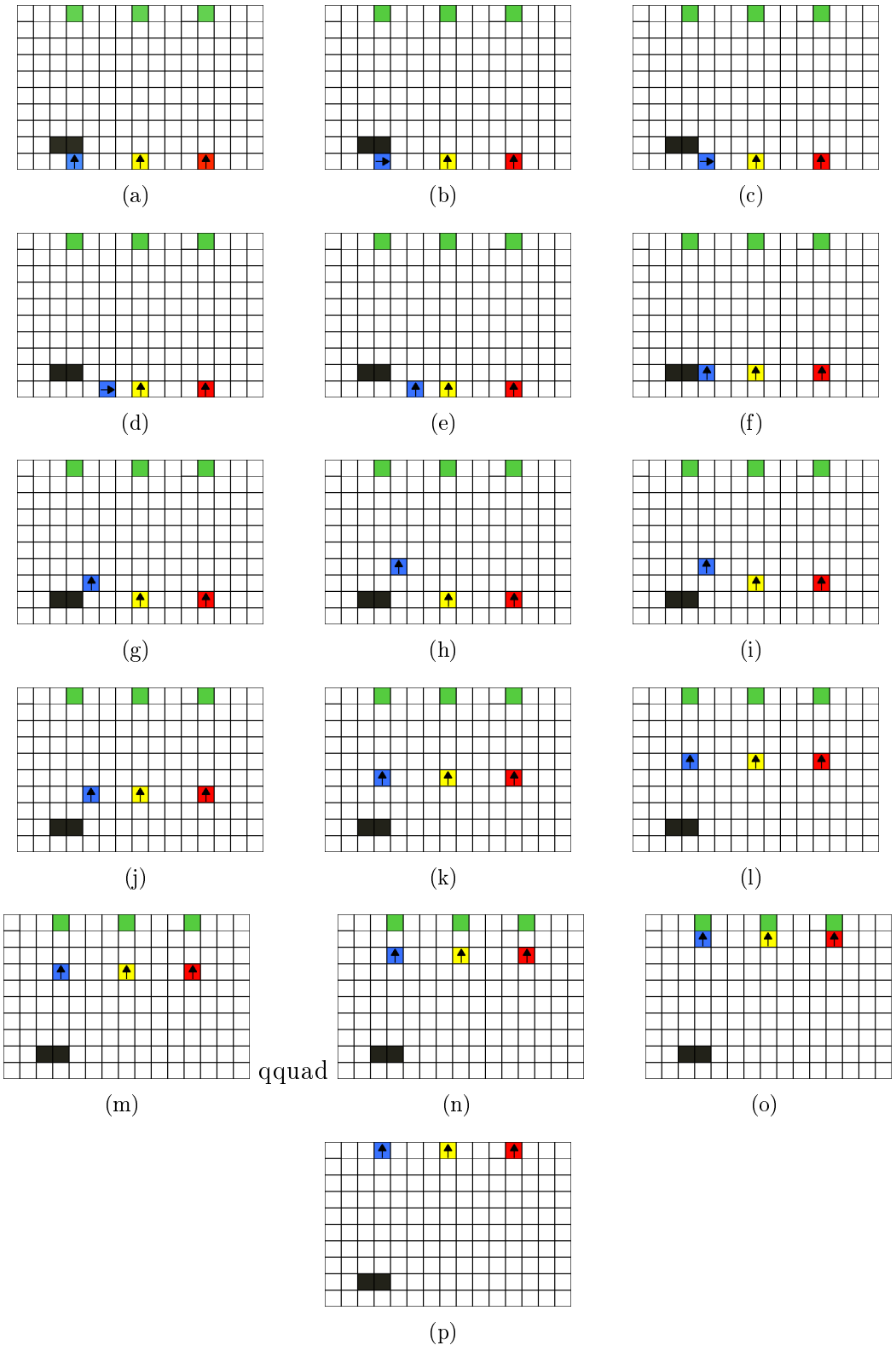


Figura 5.31: Exemplo utilizando o algoritmo sem a troca de posições.

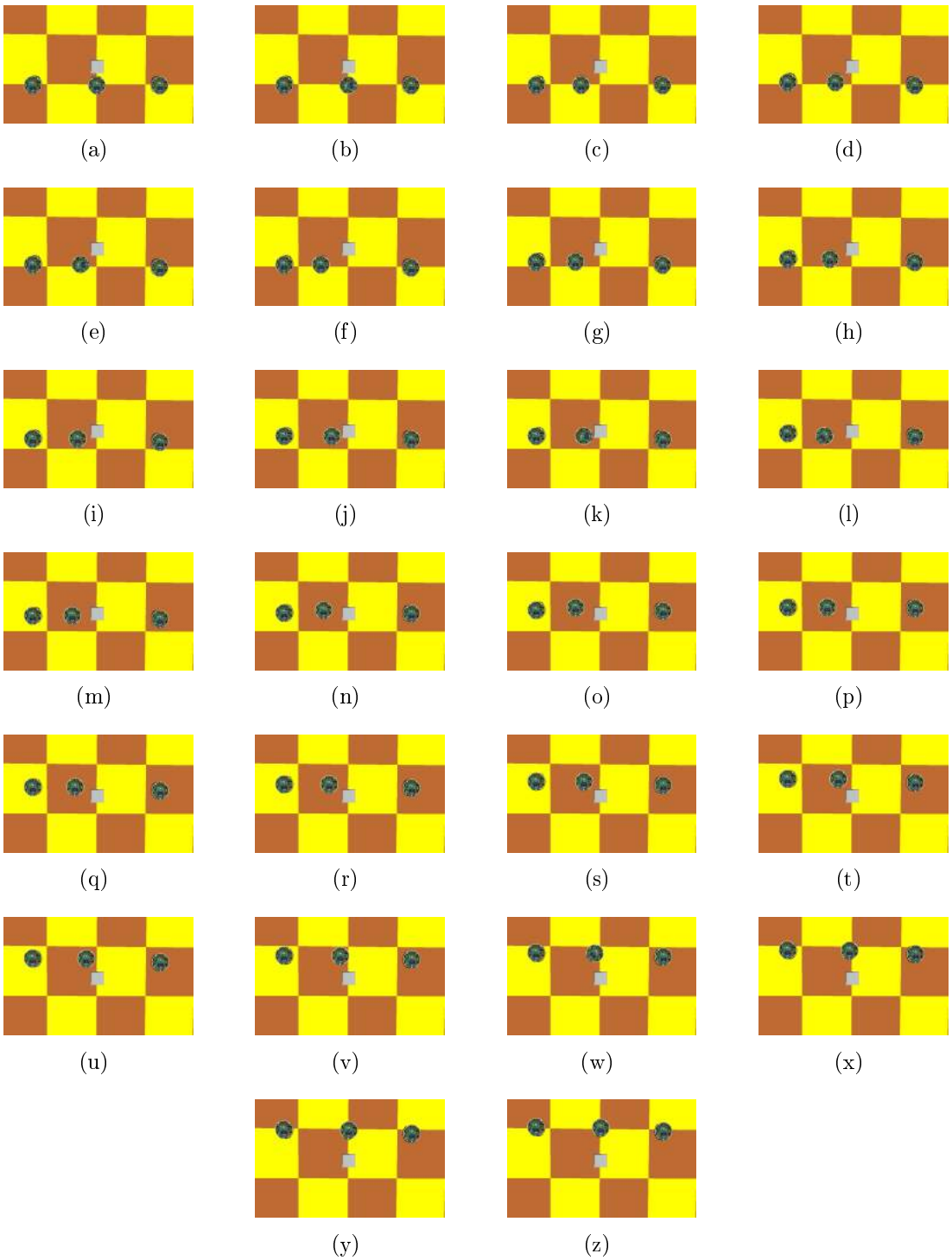


Figura 5.32: Exemplo aplicado a um time de robôs no simulador Webots do modelo utilizando a segunda modificação das regras para um único robô.

Capítulo 6

Conclusões e Trabalhos Futuros

O planejamento de caminhos para robôs móveis é um problema bastante relevante para a navegação autônoma de robôs, sendo que diversas técnicas vêm sendo pesquisadas para a resolução do mesmo. Este trabalho investigou métodos de planejamento de caminhos baseados em autômatos celulares. Em um primeiro momento, os trabalhos encontrados na literatura foram agrupados em seis abordagens distintas, escolhidas devido à características semelhantes dos métodos. Esta primeira etapa foi necessária pois existia uma escassez de trabalhos que fizessem uma análise comparativa dos métodos baseados em ACs para o planejamento de caminhos, além de viabilizar a escolha dos métodos mais promissores para a implementação em simulação e em robôs reais. Dois trabalhos de duas abordagens distintas foram estudados, implementados e analisados criticamente [Behring et al. 2000], [Ioannidis et al. 2008].

O primeiro trabalho [Behring et al. 2000] consiste em criar uma lista de passos que levará o robô da posição inicial até a meta, evitando a colisão com obstáculos estáticos, a partir de uma imagem capturada do ambiente de navegação. Para atingir este objetivo, o algoritmo utiliza dois autômatos celulares em duas fases consecutivas. O primeiro AC é utilizado com o objetivo de crescer as bordas dos obstáculos para evitar colisões do robô quando este se movimentar pelo ambiente utilizando as restrições da cinemática e dinâmica. O segundo AC é utilizado para calcular a distância entre as células livres do reticulado e a meta, até que seja calculado o valor de distância da célula contendo a posição inicial. Tendo estas distâncias, aplica-se um gradiente descendente partindo da posição inicial até encontrar a célula objetivo, sendo este o caminho a ser percorrido. Quando implementado no ambiente de simulação V-REP e utilizando a arquitetura e-puck, este primeiro método resultou em dois problemas que evitavam que o robô encontrasse uma boa solução do planejamento. O primeiro problema consistia no fato da trajetória real percorrida pelo robô ser muito diferente da projetada pelo algoritmo, devido às restrições de dinâmica simuladas pelo ambiente. Como o algoritmo não previa qualquer tipo de correção durante a navegação, o robô estacionava distante da meta ao final do processo de movimentação. Este problema foi atacado através da utilização de outra imagem do

ambiente a cada n passos para se realizar um recálculo do caminho do robô partindo da sua posição corrente. O segundo problema foi relativo a situações em que o robô se encontra inicialmente em uma área alargada de obstáculos, que impede que ele encontre um caminho válido até a meta. A abordagem utilizada para resolver este problema foi executar uma segunda difusão da distância nas células pertencentes à área dos obstáculos alargados, usando uma métrica diferente da utilizada para as células livres na primeira difusão. Com essa nova métrica, o robô se movimenta dentro da área de obstáculos alargados, indo em direção do espaço livre mais próximo, mas passando o mais distante o possível de um obstáculo real. A partir das duas principais modificações (recálculo a cada n passos e difusão das distâncias nas áreas alargadas), um novo modelo baseado em autômatos celulares para o planejamento de caminhos foi gerado. Nas simulações com o ambiente V-REP, o novo modelo se mostrou mais eficiente que seu precursor. Contudo, este modelo não foi de fato aplicado em um ambiente com robôs reais devido à falta de um sistema de captura, tratamento e envio de imagens para os robôs.

O segundo trabalho investigado [Ioannidis et al. 2008] foi escolhido pelo fato do robô utilizar seus sensores de proximidade para construir sua vizinhança a cada passo de tempo, fazendo assim uma escolha local sobre qual direção se movimentar a seguir. Além disso, o segundo trabalho também contém uma característica cooperativa, pois os robôs devem se movimentar de forma coordenada em uma formação pré-determinada. Para isto, o modelo é formado por dois conjuntos de regras baseadas em autômatos celulares: as regras de desvio de obstáculos e as regras de controle de formação. Quando o robô encontra um obstáculo em sua vizinhança, ele utilizará alguma regra de desvio sem se preocupar com o restante do time e sua formação. Quando não existir obstáculo na vizinhança, o robô irá aplicar alguma regra de controle de formação para que retorne ao seu eixo de deslocamento original (caso tenha se desviado) e o movimento coordenado dos robôs volte a ser executado mantendo a formação original. O modelo original descrito em [Ioannidis et al. 2008] foi inicialmente adaptado para a navegação de um único robô, uma vez que no artigo o ambiente é descrito prevendo um time com pelo menos 3 robôs. Com a simplificação de um único robô no ambiente, a formação desejada reside em fazer com que ele caminhe em seu eixo original, desviando-se de obstáculos eventuais e retornando ao eixo o mais rápido o possível. Na primeira simulação realizada no software Webots, o modelo não foi capaz de realizar esta tarefa, pois sempre que ele tentava desviar de um obstáculo ocorria o problema que denominamos *deadlock* da quina do obstáculo. Esse problema caracteriza-se pela dificuldade dos sensores laterais do robô reconhecerem o final do obstáculo (a quina), fazendo que o robô entre em um estado de rotação contínua. Embora esse problema não tenha sido descrito em [Ioannidis et al. 2008], ele é previsível de ocorrer em 100% de cenários com um único robô e bastante provável de ocorrer mesmo em cenários com um time de robôs.

Assim, foi realizada uma modificação no modelo, incluindo um novo estado denomi-

nado Robô_Rotacionado, que faz com que o robô dê mais um passo sempre que identifica uma quina de obstáculo e um novo conjunto de regras de desvio de obstáculos para lidar com o novo estado. Esta modificação possibilitou que o robô de fato encontrasse um caminho que desviasse dos obstáculos e retornasse à sua coluna original, resolvendo assim o problema do *deadlock*. Todavia, o robô ainda realizava várias rotações para escapar desta quina do obstáculo, o que era custoso para o planejamento. Dessa forma, foi efetuada a segunda melhoria no modelo, onde um novo conjunto de estados Robô_Rotacionado_i (com i variando de 0 a 3) foi incorporado, assim como as regras de desvio de obstáculos necessárias para lidar com esses quatro novos estados. O objetivo dessa modificação é fazer o robô se movimentar por uma distância pelo menos igual ao seu raio sempre que ele encontra uma quina de obstáculo. Através de novas simulações no ambiente Webots, foi possível observar que o novo modelo superava o problema de *deadlock* da quina com um número menor de rotações, tornando o comportamento do robô mais eficiente. A terceira e última modificação do modelo foi realizada pela alteração do conjunto de regras de controle de formação, com a inserção de um novo estado Robô_Alinhado, que faz o robô dar um passo a frente antes de se movimentar na diagonal para retornar ao seu eixo de deslocamento original. Com esta modificação, o comportamento do robô para retornar ao seu eixo original após um desvio de obstáculo é mais eficiente, pela diminuição de movimentos em ziguezague ao percorrer a lateral do obstáculo.

Com as três modificações incorporadas, um novo modelo de planejamento de caminhos baseado em regras de atualização locais de ACs foi obtido, sendo voltado a cenários com um único robô no ambiente. Esse modelo foi inicialmente avaliado no ambiente de simulação Webots, mostrando-se mais eficiente que o modelo original de [Ioannidis et al. 2008]. Posteriormente, foram realizados experimentos com um robô e-puck real, que corroboraram as análises feitas no ambiente simulado, confirmando a necessidade de correção do problema do *deadlock* da quina no modelo original, além de mostrar que as modificações efetuadas tornaram o modelo mais eficiente no desvio de obstáculos.

Numa última etapa do trabalho, foi implementado o modelo cooperativo original proposto em [Ioannidis et al. 2008] para cenários onde existem três robôs no ambiente e o time deve navegar pelo ambiente buscando manter a formação em linha reta. Para isso, o modelo original prevê uma estrutura mestre-escravo, na qual o robô central é o mestre e os demais são escravos. Além disso, os robôs devem trocar mensagens informando suas posições para que haja um movimento coordenado entre eles. Para manter a formação, os robôs utilizam a estratégia de troca de posições na formação, se deslocando em diagonal sempre que essa troca ocorre. Após a análise do modelo, foi possível identificar que, apesar de ser eficiente em retornar os robôs às suas colunas originais o mais rápido possível, a estratégia de troca de colunas aumenta a probabilidade de ocorrência de um problema que denominamos de pareamento de robôs, no qual dois robôs que estão tentando trocar de posições ficam lado-a-lado por tempo indeterminado, impossibilitando que a meta seja

de fato encontrada. Para lidar com esse problema identificado nos cenários com times de robôs, foi realizada uma alteração nas regras de controle de formação, eliminando a estratégia de troca de posições na formação. Na nova estratégia, após o desvio de um obstáculo, cada robô deve retornar ao seu próprio eixo original com movimentos em diagonal, de forma similar ao modelo para um único robô no ambiente, ainda mantendo uma estrutura mestre-escravo para controlar o deslocamento em linha dos robôs. O novo modelo cooperativo gerado com essa modificação retornou um comportamento mais eficiente nos testes realizados em um ambiente simulado, possibilitando que os robôs encontrassem um caminho até a meta, mantendo a formação sempre que possível.

Portanto, ao final do desenvolvimento da dissertação, foram propostos três novos modelos baseados em autômatos celulares para o planejamento de caminhos em robôs autônomos:

- um modelo baseado em difusão de distâncias, para um cálculo de rota livre de obstáculos, a ser percorrida por um único robô de um ponto inicial à meta, corrigido a cada intervalo de tempo, a partir de imagens capturadas do ambiente;
- um modelo baseado na identificação de obstáculos da vizinhança em tempo real e que utiliza regras locais de atualização, para realizar a navegação de um único robô buscando-se manter um eixo de deslocamento, do qual o robô se desloca apenas durante o desvio de eventuais obstáculos;
- um modelo cooperativo baseado na identificação de obstáculos da vizinhança em tempo real e que utiliza regras locais de atualização, para realizar a navegação de um time de robôs buscando-se manter a formação em linha reta;

Os resultados do primeiro e do terceiro modelo foram validados em ambientes de simulação, sendo o software V-REP utilizado no primeiro e o software Webots utilizado no terceiro. Os resultados do segundo modelo foram validados tanto através de simulações no software Webots, quanto com experimentos envolvendo robôs e-puck reais.

6.1 Trabalhos Futuros

O trabalho desenvolvido nessa dissertação focou na investigação e melhoria de dois modelos propostos anteriormente na literatura. Por isso, as modificações efetuadas nos modelos, se relacionaram a aspectos identificados na simulação dos modelos originais. Entretanto, uma vez iniciadas essas investigações em duas abordagens bem distintas, porém ambas apoiadas no uso de modelos de autômatos celulares, o trabalho pode ser continuado de várias formas e até mesmo investigando novas proposições que se afastem dos princípios dos modelos originais.

Como trabalhos futuros, vislumbramos algumas continuidades do trabalho que seriam interessantes:

- Em relação ao modelo baseado na difusão das distâncias: (i) melhorias implementação da dinâmica do robô, voltadas ao uso da odometria como foi feito no segundo modelo; (ii) implementação de um sistema onde as imagens do ambiente possam ser adquiridas, processadas e transmitidas para os robôs reais a cada passo de tempo, viabilizando assim realizar os experimentos com robôs reais; tal implementação demandaria uma interação interessante com um grupo de pesquisa de processamento digital de imagens; (iii) aplicação e adaptação do modelo para cenários com obstáculos móveis, podendo se tratar inclusive de outros robôs; o recálculo da rota a cada intervalo de tempo tem como decorrência a facilidade de adaptar o modelo para obstáculos móveis; (iv) utilização de computação evolutiva para o ajuste de parâmetros do modelo, como o valor de n e os parâmetros relacionados à dinâmica do robô;
- Em relação ao modelo baseado em regras de atualização locais: (i) ajuste nos parâmetros de odometria para melhoria do desempenho dos robôs nos experimentos reais; (ii) realização de experimentos com robôs reais do modelo cooperativo é o próximo passo sugerido para esta pesquisa, assim seria comprovada a real eficiência das modificações propostas ao modelo cooperativo; para isso, é necessária a implementação da comunicação via *Bluetooth* entre os robôs reais; (iii) utilização de algoritmos evolutivos para ajuste de parâmetros do modelo, por exemplo, o tamanho da célula, o número de passos de deslocamento na quina, os limites dos valores de leitura dos sensores, etc; (iv) utilização de algoritmos evolutivos para encontrar regras interessantes para o planejamento, sendo que por exemplo poderia ser evoluída a quantidade de estados necessários para a movimentação do robô; (v) realização de algum tipo de comunicação implícita, evitando a troca de mensagens via *Bluetooth*.

Referências Bibliográficas

[Akbarimajd e Hassanzadeh 2011] Akbarimajd, A. e Hassanzadeh, A. (2011). A Novel Cellular Automata Based Real Time Path Planning Method for Mobile Robots. *Int. Journal of Engineering Research and Applications*, 1(4):1262–1267.

[Akbarimajd e Hassanzadeh 2012] Akbarimajd, A. e Hassanzadeh, A. (2012). Autonomously Implemented Versatile Path Planning for Mobile Robots Based on Cellular Automata and Ant Colony. *International Journal of Computational Intelligence Systems*, 5(1):39–52.

[Akbarimajd e Lucas 2006] Akbarimajd, A. e Lucas, C. (2006). A New Architecture to Execute CAs-Based Path-Planning Algorithm in Mobile Robots. *2006 IEEE International Conference on Mechatronics*, pp. 478–482.

[Arkin 1998] Arkin, R. C. (1998). *Behavior-based robotics*. MIT press.

[Aurenhammer 1991] Aurenhammer, F. (1991). Voronoi diagrams - a survey of a fundamental geometric data structure. *ACM Computing Surveys (CSUR)*, 23(3):345–405.

[Avadhanula et al.] Avadhanula, V. K., Lingala, N. M., Chandrasekaran, P., Mada, V., Luo, R., e Namireddy, H. R. Navigation and path planning for robotics. http://www2.cs.lamar.edu/faculty/disrael/COSC5100/Navigationandpathplanningforrobotics.pdf

[Barraquand et al. 2000] Barraquand, J., Kavraki, L., Motwani, R., Latombe, J.-C., Li, T.-Y., e Raghavan, P. (2000). A random sampling scheme for path planning. In *Robotics Research*, pp. 249–264. Springer.

[Barraquand et al. 1992] Barraquand, J., Langlois, B., e Latombe, J.-C. (1992). Numerical potential field techniques for robot path planning. *Systems, Man and Cybernetics, IEEE Transactions on*, 22(2):224–241.

[Behring et al. 2000] Behring, C., Bracho, M., Castro, M., e Moreno, J. A. (2000). An Algorithm for Robot Path Planning with Cellular Automata. In *Proceedings of the Fourth International Conference on Cellular Automata for Research and Industry: Theoretical and Practical Issues on Cellular Automata*, pp. 11 – 19.

[Cai et al. 2007] Cai, C., Yang, C., Zhu, Q., e Liang, Y. (2007). A fuzzy-based collision avoidance approach for multi-robot systems. In *Robotics and Biomimetics, 2007. ROBIO 2007. IEEE International Conference on*, pp. 1012–1017.

[Capi et al. 2008] Capi, G., Pojani, G., e Kaneko, S.-I. (2008). Evolution of Task Switching Behaviors in Real Mobile Robots. In *Innovative Computing Information and Control, 2008. ICICIC '08. 3rd International Conference on*, pp. 495–495.

[CM-Labs 2013] CM-Labs (2013). CM Labs Simulations. http://www.vxsim.com.

[Coumans 2013] Coumans, E., e. a. (2013). Real-Time Physics Simulation. http://bulletphysics.org/wordpress/.

[Cyberbotics 2013] Cyberbotics (2013). Webots 7: robot simulator. http://www.cyberbotics.com/overview.

[De Souza 2008] De Souza, S. C. B. (2008). Planejamento de trajetória para um robô móvel com duas rodas utilizando um algoritmo A-Estrela modificado. Master's thesis, Programa de Pós-graduação em Engenharia Elétrica, COPPE/UFRJ.

[Dowling 1995] Dowling, K. (1995). Robotics: comp.robotics Frequently Asked Question. http://www.frc.ri.cmu.edu/robotics-faq.

[Gardner 1970] Gardner, M. (1970). Mathematical games: The fantastic combinations of John Conway's new solitaire game 'Life'. *Scientific American*, 223(4):120–123.

[Greeff e Nolfi 2010] Greeff, J. e Nolfi, S. (2010). Evolution of Implicit and Explicit Communication in Mobile Robots. In Nolfi, S. e Mirolli, M. (editores), *Evolution of Communication and Language in Embodied Agents*, pp. 179–214. Springer Berlin Heidelberg.

[Hwang e Ahuja 1992] Hwang, Y. K. e Ahuja, N. (1992). Gross motion planning:a survey. *ACM Computing Surveys (CSUR)*, 24(3):219–291.

[Ioannidis et al. 2011a] Ioannidis, K., Sirakoulis, G., e Andreadis, I. (2011a). Cellular ants: A method to create collision free trajectories for a cooperative robot team. *Robotics and Autonomous Systems*, 59(2):113–127.

[Ioannidis et al. 2008] Ioannidis, K., Sirakoulis, G. C., e Andreadis, I. (2008). A Cellular Automaton Collision-Free Path Planner Suitable for Cooperative Robots. *2008 Panhellenic Conference on Informatics*, pp. 256–260.

[Ioannidis et al. 2011b] Ioannidis, K., Sirakoulis, G. C., e Andreadis, I. (2011b). A Path Planning Method Based on Cellular Automata for Cooperative Robots. *Applied Artificial Intelligence*, 25(8):721–745.

[Jianjun et al. 2013] Jianjun, Y., Hongwei, D., Guanwei, W., e Lu, Z. (2013). Research about local path planning of moving robot based on improved artificial potential field. In *Control and Decision Conference (CCDC), 2013 25th Chinese*, pp. 2861–2865. IEEE.

[Junior 2008] Junior, P. L. J. D. (2008). Odometria Visual. http://www.verlab.dcc.ufmg.br/cursos/visao/2008-1/grupo01/index.

[Kari 2005] Kari, J. (2005). Theory of cellular automata: A survey. *Theoretical Computer Science*, 334:3 – 33.

[Kavraki et al. 1996] Kavraki, L. E., Svestka, P., Latombe, J.-C., e Overmars, M. H. (1996). Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *Robotics and Automation, IEEE Transactions on*, 12(4):566–580.

[Kosecka 2013] Kosecka, J. (2013). Potential Field Methods. http://cs.gmu.edu/ kosecka/cs685/cs685-potential-fields.pdf.

[Kostavelis et al. 2012] Kostavelis, I., Boukas, E., Nalpantidis, L., e Gasteratos, A. (2012). Path Tracing on Polar Depth Maps for Robot Navigation. *Cellular Automata*, pp. 395–404.

[Latombe 1991] Latombe, J. (1991). *Robot motion planning*. Kluwer Academic publishers, London.

[Lingelbach 2004] Lingelbach, F. (2004). Path planning using probabilistic cell decomposition. In *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, volume 1, pp. 467–472. IEEE.

[Marchese 1996] Marchese, F. (1996). Cellular automata in robot path planning. *Advanced Mobile Robot, 1996., Proceedings of the First Euromicro Workshop on Advanced Mobile Robots EUROBOT 96*.

[Marchese 2005] Marchese, F. (2005). A reactive planner for mobile robots with generic shapes and kinematics on variable terrains. In *Advanced Robotics, 2005. ICAR '05. Proceedings., 12th International Conference on*, pp. 23 –30.

[Marchese 2008] Marchese, F. (2008). Spatiotemporal MCA Approach for the Motion Coordination of Heterogeneous MRS. *Recent Advances in Multi Robot Systems*.

[Marchese 2011] Marchese, F. (2011). Time-invariant Motion Planner in discretized C-Spacetime for MRS. *Multi-Robot Systems, Trends and Development, Toshiyuki Yasuda*.

[Marchese 2002] Marchese, F. M. (2002). A directional diffusion algorithm on cellular automata for robot path-planning. *Future Generation Computer Systems*, 18(7):983–994.

[Mitchell 1996] Mitchell, M. (1996). Computation in cellular automata: A selected review. *Non-standard Computation*, pp. 385–390.

[Mondada et al. 2009] Mondada, F., Bonani, M., Raemy, X., Pugh, J., Cianci, C., Klaptocz, A., Magnenat, S., christophe Zufferey, J., Floreano, D., e Martinoli, A. (2009). The e-puck, a robot designed for education in engineering. In *In Proceedings of the 9th Conference on Autonomous Robot Systems and Competitions*, pp. 59–65.

[NVidia 2013] NVidia (2013). PhysX GeForce. http://www.geforce.com/hardware/technology/physx

[Oliveira 2003] Oliveira, G. M. B. (2003). Autômatos Celulares: aspectos dinâmicos e computacionais. In *Ricardo de Oliveira Anido e Paulo César Masiero. (Org.). III Jornada de Mini-cursos em Inteligência Artificial (MCIA)*, volume 8, pp. 297–345. Sociedade Brasileira de Computação.

[Parker et al. 2003] Parker, L. E., Birch, B., e Reardon, C. (2003). Indoor Target Intercept Using an Acoustic Sensor Network and Dual Wavefront Path Planning. In *In Proceedings of IEEE International Symposium on Intelligent Robots and Systems (IROS 03*, pp. 278–283.

[Perez 2009] Perez, A. (2009). Robótica Evolutiva. In *IX Escola Regional de Computação Bahia-Alagoas-Sergipe (ERBASE)*.

[Pini et al. 2007] Pini, G., Tuci, E., e Dorigo, M. (2007). Evolution of social and individual learning in autonomous robots. In *Ecal Workshop: Social Learning in Embodied Agents*.

[PUC-Rio 2013] PUC-Rio (2013). The Programming Language Lua. http://www.lua.org.

[Ramer et al. 2013] Ramer, C., Reitelshofer, S., e Franke, J. (2013). A robot motion planner for 6-DOF industrial robots based on the cell decomposition of the workspace. In *Robotics (ISR), 2013 44th International Symposium on*, pp. 1–4. IEEE.

[Robotics 2013] Robotics, C. (2013). V-Rep, Virtual Robot Experimentation Platform. http://www.coppeliarobotics.com.

[Rosenberg 2007] Rosenberg, A. (2007). Cellular ANTomata. *Parallel and Distributed Processing and Applications*, pp. 78–90.

[Rosenberg 2010] Rosenberg, A. (2010). Ants in parking lots. *Euro-Par 2010-Parallel Processing*, pp. 1–17.

[Rosenberg 2008] Rosenberg, A. L. (2008). Cellular ANTomata: Food-Finding and Maze-Threading. *2008 37th International Conference on Parallel Processing*, pp. 528–535.

[Rosenberg 2012] Rosenberg, A. L. (2012). Cellular Antomata. *Advances in Complex Systems*, 15(06):1250070.

[Schmidt e Fey 2010] Schmidt, M. e Fey, D. (2010). An Optimized FPGA Implementation for a Parallel Path Planning Algorithm Based on Marching Pixels. In *Reconfigurable Computing and FPGAs (ReConFig), 2010 International Conference on*, pp. 442–447.

[Schouwenaars et al. 2001] Schouwenaars, T., De Moor, B., Feron, E., e How, J. (2001). Mixed integer programming for multi-vehicle path planning. In *European Control Conference*, pp. 2603–2608. Citeseer.

[Shu e Buxton 1995] Shu, C. e Buxton, H. (1995). Parallel path planning on the distributed array processor. *Parallel Computing*, 21(11):1749–1767.

[Smith 2013] Smith, R., e. a. (2013). Open Dynamics Engine. http://www.ode.org.

[Soofiyani et al. 2010] Soofiyani, F. R., Rahmani, A. M., e Mohsenzadeh, M. (2010). A Straight Moving Path Planner for Mobile Robots in Static Environments Using Cellular Automata. *2010 2nd International Conference on Computational Intelligence, Communication Systems and Networks*, pp. 67–71.

[Tavakoli et al. 2008] Tavakoli, Y., Javadi, H. H. S., e Adabi, S. (2008). A cellular automata based algorithm for path planning in multi-agent systems with a common goal. *International Journal of Computer Science and Network Security. v8*, pp. 119–123.

[Tikhanoff et al. 2008] Tikhanoff, V., Cangelosi, A., Fitzpatrick, P., Metta, G., Natale, L., e Nori, F. (2008). An Open-source Simulator for Cognitive Robotics Research: The Prototype of the iCub Humanoid Robot Simulator. In *Proceedings of the 8th Workshop on Performance Metrics for Intelligent Systems*, PerMIS '08, pp. 57–61, Gaithersburg, Maryland, New York, NY, USA. ACM.

[Tzionas et al. 1997] Tzionas, P., Thanailakis, a., e Tsalides, P. (1997). Collision-free path planning for a diamond-shaped robot using two-dimensional cellular automata. *IEEE Transactions on Robotics and Automation*, 13(2):237–250.

[Unbehauen 2009] Unbehauen, H. (2009). *Control Systems, Robotics and Automation*. Eolss Publishers Company Limited.

[Weisstein 2013] Weisstein, E. W. (2013). Piano Mover's Problem. From MathWorld–A Wolfram Web Resource. http://mathworld.wolfram.com/PianoMoversProblem.html.

[Wolfram 1983] Wolfram, S. (1983). Statistical mechanics of cellular automata. *Reviews of modern physics*, 55(3):601–644.

[Wuensche e Lesser 1992] Wuensche, A. e Lesser, M. (1992). *The global dynamics of cellular automata: An atlas of basin of attraction fields of one-dimensional cellular automata*, volume 1. Addison Wesley publishing company.

[Zelinsky et al. 1993] Zelinsky, A., Jarvis, R., Byrne, J., e Yuta, S. (1993). Planning paths of complete coverage of an un- structured environment by a mobile robot. In *Proceedings of International Conference on Advanced Robotics*, pp. 533–538.

[Zhang et al. 2013] Zhang, Y., Fattahi, N., e Li, W. (2013). Probabilistic roadmap with self-learning for path planning of a mobile robot in a dynamic and unstructured environment. In *Mechatronics and Automation (ICMA), 2013 IEEE International Conference on*, pp. 1074–1079. IEEE.

[Zieliński e Winiarski 2010] Zieliński, C. e Winiarski, T. (2010). General specification of multi-robot control system structures. *Bulletin of the Polish Academy of Sciences: Technical Sciences*, 58(1):15–28.

[École Polytechnique Fédérale de Lausanne EPFL 2013] École Polytechnique Fédérale de Lausanne EPFL (2013). E-Puck Education Robot. http://www.e-puck.org.