UNIVERSIDADE FEDERAL DE SANTA CATARINA
CAMPUS ARARANGUÁ

Thiago Zanivan Felisberto

ROBÔ SOLUCIONADOR DE SUDOKU

Araranguá, dezembro de 2015.

Thiago Zanivan Felisberto

ROBÔ SOLUCIONADOR DE SUDOKU

Trabalho de Conclusão de Curso submetido à Universidade Federal de Santa Catarina, como parte dos requisitos necessários para a obtenção do Grau de Bacharel em Engenharia de Computação.

Araranguá, dezembro de 2015.

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Felisberto, Thiago Zanivan
Robô Solucionador de Sudoku / Thiago Zanivan Felisberto ; orientador, Fabrício de Oliveira Ourique - Araranguá, SC, 2015.
144 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Campus Araranguá. Graduação em Engenharia de Computação.

Inclui referências

1. Engenharia de Computação. 2. sudoku. 3. visão de máquina. 4. processamento digital de imagens. 5. robótica. I. Ourique, Fabrício de Oliveira. II. Universidade Federal de Santa Catarina. Graduação em Engenharia de Computação. III. Título.

Thiago Zanivan Felisberto

ROBÔ SOLUCIONADOR DE SUDOKU

Este Trabalho de Conclusão de Curso foi julgado aprovado para a obtenção do Título de "Bacharel em Engenharia de Computação", e aprovado em sua forma final pelo Curso de Engenharia de Computação.

Araranguá, dezembro de 2015.

Professor Anderson Luiz Fernandes Perez, Dr.
Coordenador do Curso

Banca Examinadora:

Professor Fabrício de Oliveira Ourique, Dr.
Orientador

Professor Anderson Luiz Fernandes Perez, Dr.
Membro

Professor Gustavo Medeiros de Araújo, Dr.
Membro

Aos meus pais, cujo apoio foi fundamental durante esta jornada.

AGRADECIMENTOS

A Deus, por ter me dado saúde e força para superar as dificuldades. A minha família e amigos, pelo carinho e incondicional apoio prestados durante todo o curso. A Universidade Federal de Santa Catarina, pelos cursos oferecidos com excelência na região de Araranguá. A todos os colegas de curso, que me motivaram a seguir na caminhada. Aos docentes do curso de Engenharia de Computação, por toda a sabedoria transmitida. Especialmente ao professor Dr. Fabrício de Oliveira Ourique, que generosamente aceitou me guiar durante este trabalho, pela infindável paciência, apoio e confiança em mim depositada durante este empreendimento. Ao professor Dr. Anderson Luiz Fernandes Perez, pelas valiosas orientações concedidas durante o desenvolvimento deste trabalho. À professora Eliane Pozzebon, pelos esforços empreendidos na supervisão deste trabalho. A todos aqueles que contribuíram de alguma forma para que a conclusão deste trabalho fosse possível.

"O lucro do nosso estudo é tornarmo-nos melhores e mais sábios."

Michel Eyquem de Montaigne

RESUMO

Os avanços tecnológicos dos dispositivos de aquisição de imagens, aliados ao crescente aprimoramento do hardware computacional, estão impulsionando o desenvolvimento dos sistemas de visão de máquina. Estes sistemas são atualmente empregados para solucionar problemas de naturezas diversas, em áreas como medicina, biologia e, em especial, na indústria. Este trabalho apresenta o projeto de um robô, construído exclusivamente com peças de LEGO, capaz de solucionar jogos de Sudoku escritos em papel. Trata-se de um plotador cartesiano controlado por um sistema de visão de máquina. O projeto visa introduzir princípios, modelos e aplicações dos sistemas de visão. O processo executado pelo robô inicia-se com a captura de uma imagem do seu espaço de trabalho, utilizando-se de uma *webcam*. A imagem capturada é então analisada pelo sistema de visão, que identifica o jogo, interpreta-o e o soluciona em memória. Finalmente, a solução é escrita pelo robô por meio de uma caneta acoplada ao seu elemento terminal.

Palavras-chave: sudoku, visão de máquina, processamento digital de imagens, robótica.

ABSTRACT

Technological advances in image acquisition devices, combined with the increasingly improvements in computer hardware, are boosting the development of machine vision systems. These systems are currently employed to solve a lot of problems, in areas such as medicine, biology and particularly in the industry. This paper presents the design of a Sudoku solver robot that was built exclusively with LEGO bricks. It consists of a plotter printer, controlled by a machine vision system. The project aims to introduce principles, models and applications of vision systems. The process executed by the robot starts capturing an image of its workspace with a webcam. The captured image is then analyzed by the vision system, which identifies the game, interprets and solves it. Finally, the solution is written by the robot with a pen that is attached to its terminal element.

Keywords: sudoku, machine vision, digital image processing, robotics.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

SUMÁRIO

1 INTRODUÇÃO

Visão computacional é a área da ciência que estuda métodos de aquisição, processamento, análise e compreensão de imagens, geralmente provenientes do mundo real, para produzir informações numéricas ou simbólicas (DAVIES, 2012). A *visão de máquina*, por sua vez, se refere a aplicação da visão computacional para tomada de decisão em processos de automação, tais como a inspeção automática de produtos e a retroalimentação para controle de movimento de robôs.

Em múltiplos aspectos, a visão de computacional constitui-se como um problema de inteligência artificial completo, onde objetiva-se estabelecer uma relação lógica sobre um conjunto de dados aparentemente desconexos (DAUGMAN, 2010). Especificamente na visão de máquina, o objetivo é gerar descrições úteis de cenas visuais a partir de sinais de imagens. O principal desafio das aplicações de visão é transformar estes sinais em informações úteis para compreensão do mundo real, permitindo que uma máquina interaja de forma inteligente com o ambiente onde está inserida.

Os sistemas de visão de máquina estão desempenhando papéis cada vez mais importantes em áreas diversas, como medicina, biologia, engenharias e, em especial, automação industrial. Isto se deve principalmente aos crescentes avanços dos dispositivos de aquisição de imagens e ao aprimoramento do hardware computacional, ao passo em que o custo destas tecnologias reduz progressivamente. Não coincidentemente, sistemas de visão para aplicações industriais formam atualmente um negócio multibilionário (CARROLL, 2015).

Frente a este cenário, o presente trabalho apresenta o projeto de um robô capaz de solucionar jogos de Sudoku impressos em papel a partir de fotografias dos mesmos. O robô é controlado por um sistema de visão de máquina, que reconhece os jogos em imagens digitais capturadas de seu espaço de trabalho, resolve-os em memória e coordena seus atuadores para escrita da solução no papel. Além disso, a estrutura do robô proposto, um plotador cartesiano, se assemelha a de máquinas utilizadas em alguns processos de automação industrial, principal segmento onde os sistemas de visão são atualmente empregados (CARROLL, 2015).

O Sudoku é um popular jogo de quebra-cabeças onde o jogador deve preencher as células vazias de uma grade 9×9 com números inteiros de 1 até 9, de forma que em todas as linhas, todas as colunas e nas nove regiões 3×3, cada dígito apareça apenas uma vez, respeitando

um conjunto de células já preenchidas chamadas de pistas.

Dentre as tarefas executadas pelo robô para resolver os jogos estão a localização da grade, a identificação de células vazias, o reconhecimento das pistas e o controle de movimento do robô. As operações realizadas para tanto utilizam fundamentos teóricos que servem como base para diversas outras aplicações práticas.

1.1 OBJETIVOS

Esta seção descreve o objetivo geral e os objetivos específicos do trabalho de conclusão de curso.

1.1.1 Objetivo Geral

Projetar um robô capaz de solucionar jogos de Sudoku impressos em papel.

1.1.2 Objetivos Específicos

1. Levantar o estado da arte em algoritmos utilizados para solucionar jogos de Sudoku.

2. Desenvolver um sistema de visão de máquina para reconhecer jogos de Sudoku em fotografias digitais.

3. Testar e validar o sistema de visão desenvolvido em (2).

4. Projetar um robô plotador cartesiano utilizando exclusivamente peças de LEGO, utilizado para escrever caracteres em folhas de papel com uma caneta acoplada ao seu elemento terminal.

5. Desenvolver um sistema de controle de movimento para o robô projetado em (4).

6. Construir o robô solucionador de Sudoku integrando os componentes obtidos como resultado dos objetivos anteriores.

1.2 JUSTIFICATIVA

Apesar de seu desenvolvimento recente, Davies (2012) esclarece que ainda há muito trabalho a realizar nos campos de visão computacional e de máquina. Mesmo alguns problemas fundamentais, como o reconhecimento de caracteres, ainda precisam ter suas soluções aperfeiçoadas para que possam ser considerados resolvidos por completo.

Este trabalho se justifica principalmente como uma maneira de introduzir princípios, modelos e aplicações dos sistemas de visão aos leitores. Os conceitos aqui utilizados para desenvolver um sistema de reconhecimento de jogos de Sudoku servem como base para diversas outras aplicações da área.

Não obstante, os fundamentos de robótica e de controle empregados na construção do robô são os mesmos que os utilizados para desenvolver máquinas amplamente empregadas na indústria. Assim, este trabalho também se apresenta como uma contribuição introdutória ao desenvolvimento de robôs desta classe.

Por fim, o problema de resolver jogos de Sudoku se caracteriza como um problema de tempo polinomial não determinístico completo (NP-completo) quando generalizado para grades de tamanho $N^2 \times N^2$ (ERCSEY-RAVASZ; TOROCZKAI, 2012). Determinar se é possível ou não resolver esta classe de problemas em tempo polinomial é uma das principais questões em aberto na ciência da computação (GAREY; JOHNSON, 1979). A pesquisa sobre métodos de solução de jogos de Sudoku pode, portanto, contribuir na área de complexidade computacional.

1.3 METODOLOGIA

O trabalho desenvolver-se-á em seis grandes etapas. A primeira etapa consistirá em uma pesquisa bibliográfica exploratória sobre jogos de Sudoku. As regras do jogo, um breve histórico e os principais algoritmos e métodos utilizados para resolver os jogos serão apresentados.

Na segunda etapa, um estudo qualitativo de técnicas de aquisição e processamento digital de imagens será conduzido. Este estudo buscará apresentar fundamentos teóricos básicos das operações realizadas pelo sistema de visão de máquina desenvolvido para reconhecer jogos de Sudoku. Técnicas de detecção de linhas e formas geométricas, segmentação e reconhecimento de caracteres, filtros e operações morfológicas são exemplos de operações discutidas.

Na terceira etapa será realizada uma pesquisa bibliográfica so-

bre configurações de robôs utilizadas atualmente. Buscar-se-á discorrer sobre conceitos básicos de robótica, criando uma base teórica para o projeto mecânico do robô solucionador de Sudoku.

A quarta etapa consistirá no projeto da estrutura mecânica do robô utilizado para escrever as soluções de jogos de Sudoku no papel.

Na quinta etapa, o sistema de visão de máquina responsável por reconhecer os jogos de Sudoku será descrito.

Finalmente, na sexta e última etapa, o sistema de controle de movimento do robô será apresentado.

1.4 ORGANIZAÇÃO DO TRABALHO

Além desta **Introdução**, o presente trabalho está organizado em mais 5 (cinco) capítulos, que abordam os seguintes conteúdos:

O **Capítulo 2** descreve as regras do Sudoku, apresenta um breve resumo histórico das origens do jogo, analisa métodos tradicionais de solução manual e apresenta os principais algoritmos computacionais utilizados para resolver jogos de Sudoku.

O **Capítulo 3** faz um breve apanhado das bases teóricas necessárias para desenvolver o sistema de visão de máquina que reconhece jogos de Sudoku. São elucidados conceitos básicos sobre processamento de imagens, técnicas de processamento empregadas no sistema e métodos de reconhecimento óptico de caracteres (OCR).

O **Capítulo 4** aborda fundamentos de robótica necessários para a construção do robô solucionador de Sudoku, tais como aspectos mecânicos e de controle.

O **Capítulo 5** apresenta o robô solucionador de Sudoku construído. O projeto mecânico do mesmo é detalhado, incluindo a análise cinemática, o dimensionamento e a sua modelagem. O sistema de controle, responsável por ler, interpretar e resolver os jogos de Sudoku, além de controlar os movimentos do robô, também é apresentado.

O **Capítulo 6** discute os resultados obtidos e indica possibilidades de melhorias para o projeto, além de realizar sugestões para trabalhos futuros.

2 SUDOKU

Sudoku é um popular jogo de quebra-cabeças baseado na colocação lógica de números em uma grade de 9×9 células. O objetivo é preencher todas as células em branco com dígitos de 1 até 9 de modo que os dígitos não se repitam em nenhuma linha, coluna ou região 3×3. Quando estendido para grades de dimensões $N^2 \times N^2$, o Sudoku se caracteriza como um exemplo clássico do problema computacional de cobertura exata, um tipo de problema de satisfação de restrições polinomial não determinístico completo (NP-completo) (ERCSEY-RAVASZ; TOROCZKAI, 2012). Determinar se é possível ou não resolver esta classe de problemas em tempo polinomial é uma das principais questões em aberto na ciência da computação (GAREY; JOHNSON, 1979).

O Sudoku exige apenas raciocínio lógico do jogador para alcançar a solução e é utilizado como forma de entretenimento pelas pessoas. Ao mesmo tempo, é objeto de estudo de matemáticos e cientistas da computação, haja vista que o desenvolvimento de novos métodos de solução podem contribuir com pesquisas do problema de cobertura exata (MCGUIRE; TUGEMANN; CIVARIO, 2012).

Este capítulo descreve as regras do jogo; apresenta um breve resumo histórico das origens do mesmo; analisa métodos tradicionais de solução manual; e, finalmente, apresenta os principais algoritmos computacionais tipicamente utilizados para resolver o problema do Sudoku.

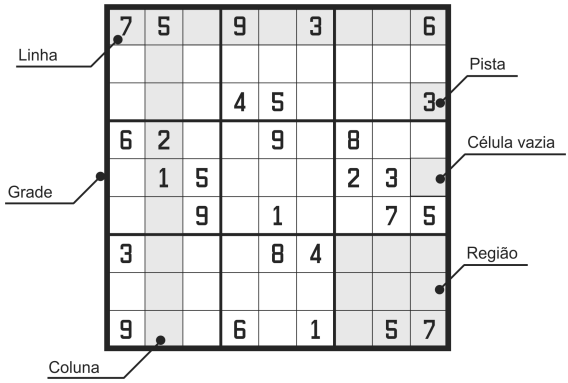
2.1 REGRAS

Sudoku é um tipo de quebra-cabeças onde o jogador deve organizar dígitos numéricos numa sequência lógica. O objetivo do jogo é preencher as *células* vazias de uma *grade* de dimensões 9×9, com números inteiros no intervalo fechado de 1 até 9, tal que cada *linha*, *coluna* e *região* 3×3 da grade contenha apenas uma ocorrência de cada número. Esta é a razão do nome *"sudoku"*, que significa *"números únicos"* em japonês (WILSON, 2006). O conjunto formado pela linha, coluna e região de uma célula é chamado de *grupo* da célula.

O jogo possui algumas *pistas* iniciais, que preenchem parcialmente a grade. Tratam-se de números inseridos em algumas células, dispostos de forma que exista uma e apenas uma solução para o problema. O número de pistas fornecidas é variável, mas para que uma única solução seja alcançada são necessárias no mínimo 17 pistas, como

provam McGuire, Tugemann e Civario (2012). A Figura 1 ilustra os elementos que compõem o jogo.

Figura 1 – Elementos do Sudoku



Fonte: Elaborada pelo autor

A Figura 2 apresenta um jogo de Sudoku e sua solução ao lado, ilustrando as regras descritas.

Figura 2 – Exemplo de sudoku e sua solução

7	5		9		3			6
			4	5				3
6	2			9		8		
	1	5				2	3	
		9		1			7	5
3				8	4			
9			6		1		5	7

(a) Jogo de sudoku

7	5	8	9	2	3	1	4	6
2	4	3	1	6	7	5	9	8
1	9	6	4	5	8	7	2	3
6	2	7	3	9	5	8	1	4
8	1	5	7	4	6	2	3	9
7	3	9	8	1	2	6	7	5
3	7	8	5	8	4	9	6	2
5	6	4	2	7	9	3	8	1
9	8	2	6	3	1	4	5	7

(b) Solução do jogo

Fonte: Elaborada pelo autor

A principal atração do jogo é que suas regras são extremamente simples, apesar de o raciocínio necessário para alcançar a solução final ser complexo em alguns casos.

Os números no Sudoku são utilizados apenas por comodidade. Qualquer combinação de símbolos distintos, tais como letras, cores, ou formas podem ser usadas no jogo sem alterar as regras. Algumas variações, usam letras, como o *Scramblets* e o *Sudoku Words*, por exemplo (GORDON; LONGO, 2006). Existem ainda variações que utilizam grades e regiões com diferentes dimensões, tais como grades de 16×16 e regiões de 4×4 (WILSON, 2006). O presente trabalho considera apenas o formato tradicional do jogo, mas todas as soluções apresentadas podem ser adaptadas e estendidas para um grande número de variações do jogo.

Os sudokus podem ser classificados em níveis de dificuldade para solução. As publicações normalmente utilizam quatro níveis distintos para classificar seus jogos (BERGGREN; NILSSON, 2012), sendo eles: *fáceis*; *intermediários*; *difíceis*; e *desafiadores*. Devido a natureza subjetiva dessa classificação, um mesmo jogo pode ser classificado em diferentes categorias, dependendo dos critérios adotados pelos editores – um jogo classificado como *fácil* em uma publicação pode ser classificado como *intermediário* em outra e assim por diante.

Surpreendentemente, Ercsey-Ravasz e Toroczkai (2012) mostram que o número de pistas dadas no jogo tem pouca ou nenhuma relação com o nível de dificuldade percebido por humanos ou algoritmos computacionais de uma forma em geral. Em alguns casos, um jogo com um número pequeno de pistas pode apresentar solução trivial, ao passo que um jogo com um número de pistas acima da média pode apresentar solução extremamente complexa. A dificuldade do quebra-cabeças tem maior relação com a relevância e o posicionamento das pistas do que com sua quantidade (ERCSEY-RAVASZ; TOROCZKAI, 2012).

2.2 RESUMO HISTÓRICO

O nome *Sudoku* foi dado ao jogo no Japão, e consiste na junção dos caracteres japoneses *"Su"*, que significa *"número"*, e *"Doku"*, que significa *"único"* (WILSON, 2006). Apesar do nome, o Sudoku não foi uma invenção japonesa. Ele possui raízes em antigos quebra-cabeças de números, em especial, nos arranjos de *quadrados mágicos* (WILSON, 2006).

Um quadrado mágico é um arranjo de números naturais distintos, em uma matriz quadrada de tamanho $N \times N$. Nele, os números em cada linha, coluna e nas diagonais principal e secundária, resultam no mesmo valor quando somados, uma característica resultante da uti-

lização de um mesmo conjunto de números que não se repetem para todas as linhas. Quadrados mágicos de quaisquer dimensões podem ser construídos, com exceção de 2×2, que é um caso impossível. O quadrado mágico 1×1 é trivial. O menor quadrado não trivial é o de dimensões 3×3 (WILSON, 2005).

É creditada ao grande matemático suíço Leonhard Euler a criação das bases do jogo que conhecemos hoje como Sudoku (WILSON, 2006). Talvez apenas como um *hobby*, Euler desenvolveu as bases do jogo, que ele chamou de *Quadrados Latinos*. Euler modificou as regras originais do quadrado mágico, limitando a quantidade de dígitos utilizados a N – tamanho de uma linha ou coluna da grade –, retirando as restrições das somas diagonais e passando a exigir dígitos distintos apenas nas linhas e nas colunas, ao invés de toda a grade.

Euler percebeu que as regras das somas nas linhas e nas colunas seriam sempre satisfeitas devido a restrição de utilização de um mesmo conjunto de N dígitos. Ele utilizou então letras como símbolos, ao invés de números, tornando o jogo um problema de análise combinatória (WILSON, 2005). Suas ideias sobre o assunto foram publicadas em 1782.

O formato atual do Sudoku foi proposto por Howard Garns, um arquiteto americano, que adicionou ao problema do quadrado latino 9×9 a restrição de números únicos nas 9 regiões de tamanho 3×3 que o compõe. O primeiro sudoku foi publicado em 1979 pela Dell Magazines, com o nome de *Number Place* (WILSON, 2006).

O jogo foi introduzido no Japão pela Nikoli, na revista Monthly Nikolist, em abril de 1984, com o nome Sudoku (WILSON, 2006). Ele logo se tornou um passa tempo muito popular no país. No entanto, o sucesso se restringiu ao Japão, e o Sudoku não conseguiu atrair a mesma atenção no ocidente até o fim de 2004. Foi quando o The Times de Londres publicou seu primeiro Sudoku, no dia 12 de novembro daquele ano (WILSON, 2006). O jogo então se tornou um fenômeno por todo o mundo.

2.3 MÉTODOS DE SOLUÇÃO

O Sudoku exige do jogador apenas lógica para ser solucionado. Apesar dos números, nenhum cálculo aritmético é necessário. A ampla maioria dos métodos de solução estão baseados em duas metodologias: *eliminação de candidatos* e *tentativa-erro* (WILSON, 2005). As seções seguintes apresentam os métodos mais populares utilizados para resolver o quebra-cabeças.

2.3.1 Eliminação de Candidatos

Na *eliminação do candidatos*, o progresso é feito através de sucessivas eliminações de números possíveis (candidatos) para uma ou mais células, de forma que reste apenas uma opção possível para a célula em análise (GORDON; LONGO, 2006). Esta metodologia garante ao jogador que o número inserido está correto e não precisará ser modificado posteriormente, ou, no pior caso, restringe a quantidade de opções possíveis para uma célula a um número menor.

São apresentadas a seguir as principais técnicas baseadas na metodologia de eliminação de candidatos utilizadas por jogadores de Sudoku.

2.3.1.1 Posição Única

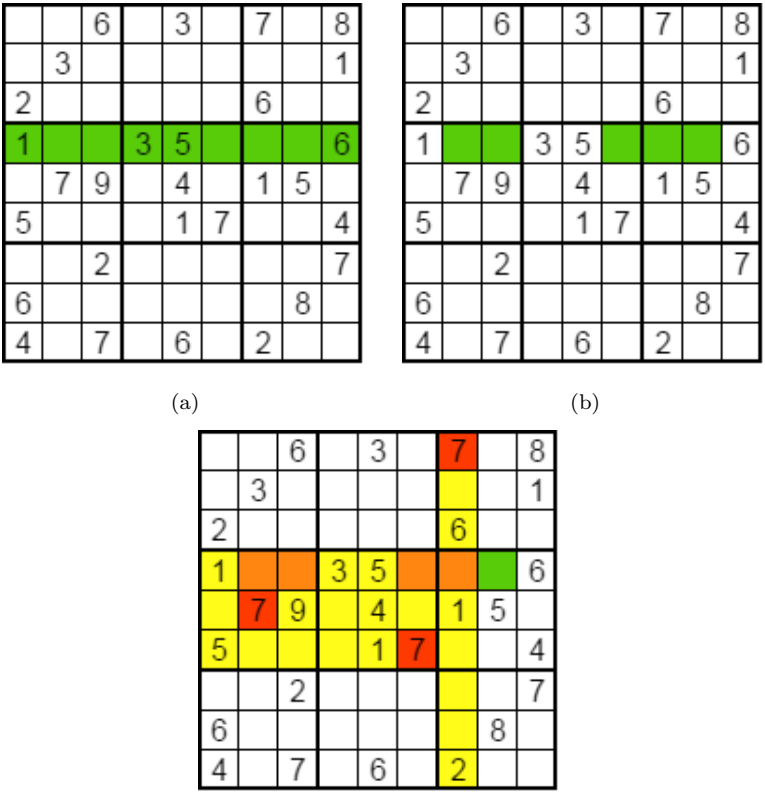
A técnica da *posição única* é uma das mais utilizadas pelos jogadores para resolver os jogos, ainda que eles não saibam que estão utilizando uma técnica. Ela consiste em escolher um grupo e então analisar todos os números que ainda não foram inseridos (GORDON; LONGO, 2006). Devido a disposição dos números na grade, as posições onde este número pode ser colocado serão limitadas.

Em muitos casos, existirão duas ou três células onde a colocação do número analisado será válida, mas em alguns casos existirá apenas um lugar possível. Se este for o caso, o jogador terá certeza de que a célula foi preenchida com o número correto, uma vez que não existirá outro lugar para colocá-lo (WILSON, 2005).

A Figura 3 destaca em (a) a linha de um jogo de Sudoku onde este raciocínio pode ser aplicado para o número 7. Sabe-se que toda linha deve conter uma ocorrência de cada número para que a solução final seja alcançada. Como o 7 não está presente nesta linha, uma das cinco células vazias deve ser preenchida com o mesmo, como mostrado em (b).

Entretanto, devido a restrição de repetição nas regiões, o 7 não pode ser colocado em nenhuma das primeiras 3 células vazias (partindo da esquerda) da linha. A quarta célula, por sua vez, já possui o 7 em sua coluna. Estas restrições são destacadas em laranja na Figura 3, item (c). Resta portanto uma posição única onde o 7 pode ser disposto, destacada em verde no mesmo item.

Figura 3 – Aplicação da técnica de posição única



(a) (b)

(c)

Fonte: Adaptada de Gordon e Longo (2006)

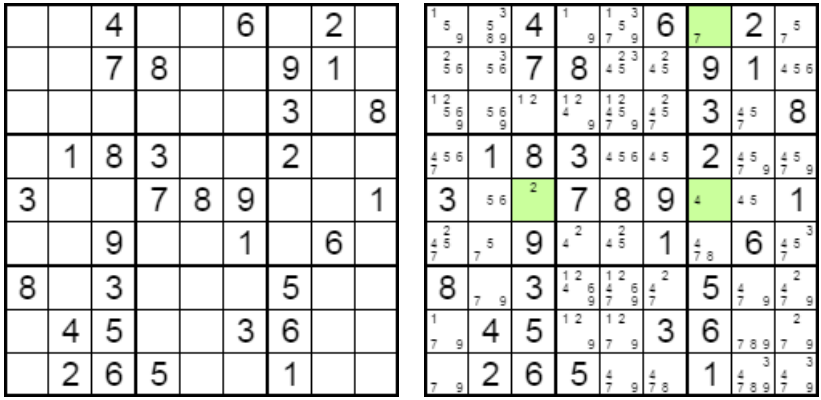
2.3.1.2 Candidato Único

A técnica de *posição única*, apresentada anteriormente, pode ser considerada um caso especial da técnica de candidato único (GORDON; LONGO, 2006). Esta técnica é uma das mais simples, especialmente se marcações de lápis forem utilizadas para marcar os candidatos de cada célula.

Basicamente, se todas as possibilidades para uma célula em particular se reduzirem a um candidato, então este número deve ser inse-

rido na célula vazia (WILSON, 2005). A Figura 4 (a) mostra um *puzzle* onde esta técnica pode ser empregada. Em (b) as possibilidades para todas as células vazias são apresentadas. É possível observar claramente que existe um candidato único para três posições, destacadas na imagem. O processo pode ser repetido quantas vezes for possível.

Figura 4 – Aplicação da técnica de candidato único



(a) (b)

Fonte: Adaptada de Gordon e Longo (2006)

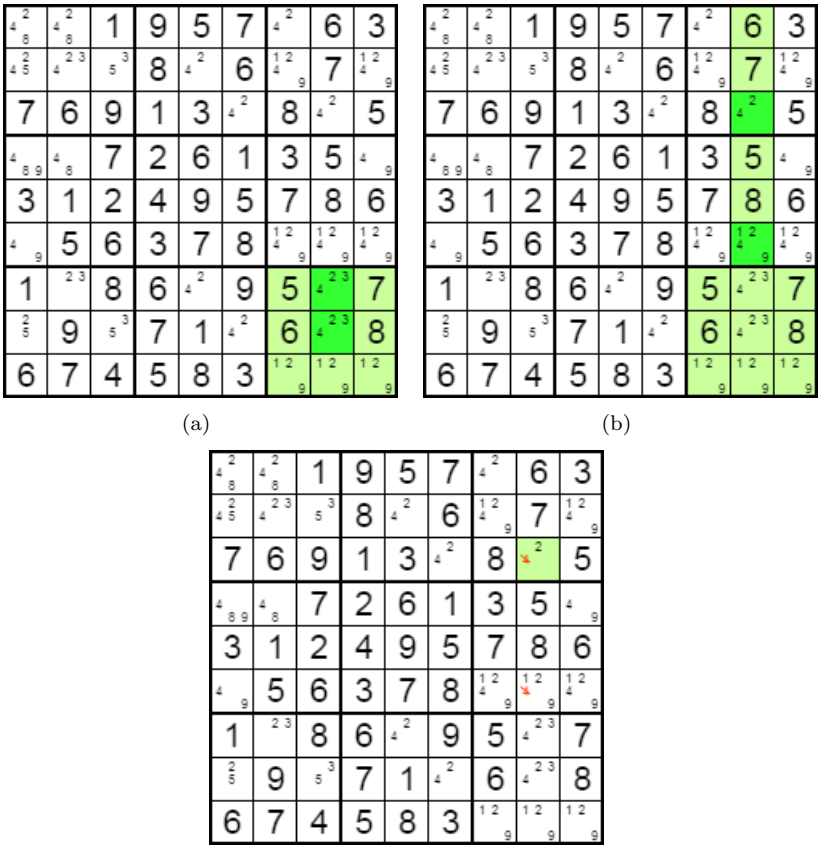
2.3.1.3 Linha Candidata

Esta técnica consiste em observar uma região e buscar por casos onde um número em particular só pode ser inserido em uma das linhas – ou uma das colunas (GORDON; LONGO, 2006). Mesmo que não fique claro em qual célula este número deve ser colocado, nenhuma das outras posições naquela linha, ou em outras duas regiões, poderão conter o número analisado. Desta forma, este candidato poderá ser removido para as células vazias desta linha nas outras regiões.

A Figura 5 destaca em (a) um caso de linha candidata. Existem apenas duas células vazias onde candidato '4' pode ser inserido, e elas estão na mesma coluna. Isso significa que o '4' deve estar na região destacada, e consequentemente em mais nenhuma célula daquela coluna. Desta forma, é possível remover o candidato '4' de outras células desta coluna, destacadas em (b). Neste exemplo, a eliminação por linha candidata deixou um único candidato, o número '2', para a célula vazia

destacada em (c), e o jogador pode preenchê-la com este valor tendo certeza de que é o número correto.

Figura 5 – Aplicação da técnica de linha candidata



(a) (b)

(c)

Fonte: Adaptada de Gordon e Longo (2006)

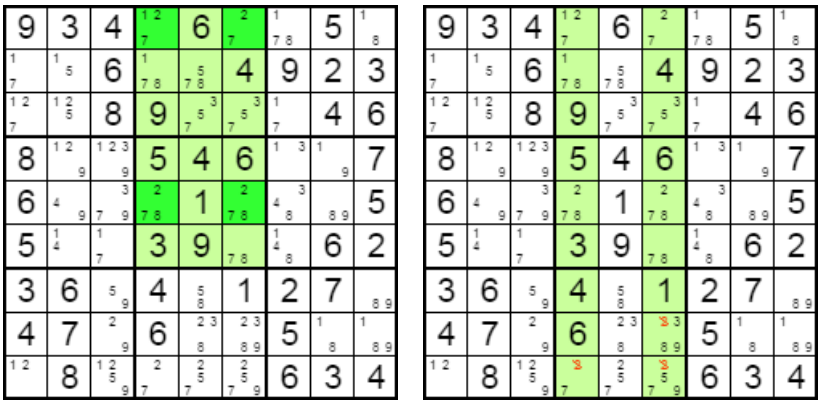
2.3.1.4 Par Duplo

A técnica de *par duplo* consiste em observar dois pares de células vazias para um valor candidato e utilizá-lo para remover candidatos de outras regiões. Ela é uma extensão da técnica *linha candidata*, apre-

sentada anteriormente.

A Figura 6 apresenta um exemplo. Nas duas regiões destacadas em (a), o candidato '2' pode estar apenas nas colunas 4 e 6. Como o '2' está limitado a essas posições nos blocos superiores, isto significa que as colunas 4 e 6 serão preenchidas com este candidato nestes blocos. Assim, o candidato '2' pode ser removido do bloco inferior se estiver em qualquer uma destas colunas – ou seja, o '2' deverá estar na coluna do meio no bloco inferior. Este procedimento é ilustrado em (b).

Figura 6 – Aplicação da técnica de par duplo



(a) (b)

Fonte: Adaptada de Gordon e Longo (2006)

2.3.1.5 Par Sozinho

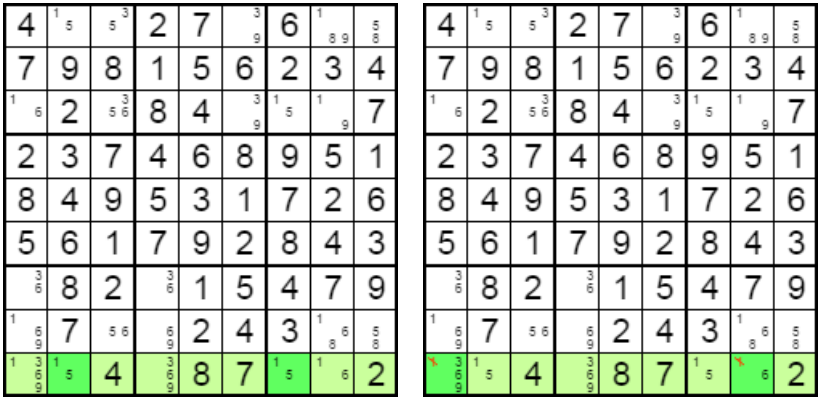
A técnica de *par sozinho* consiste na observação de que se um mesmo par de números sozinhos for candidato em duas células de uma linha, coluna ou região, isto significa que esses candidatos devem necessariamente aparecer nessas duas células (GORDON; LONGO, 2006). O mesmo raciocínio pode ser estendido a trios e quartetos de candidatos-células.

A Figura 7 exemplifica a aplicação da técnica. Em (a) é possível observar que um par sozinho, composto pelos números '1' e '5', é candidato nas duas células destacadas da última linha do jogo. Não é possível saber qual das posições deve ser preenchida com '1' e qual deve ser preenchida com '5', mas existe a certeza de que os dois candidatos

só poderão ser inseridos nestas células.

Pode-se inferir, portanto, que nenhuma outra célula vazia desta linha poderá conter estes candidatos. Logo, estes candidatos podem ser removidos de todas as demais células da linha, como mostrado em (b). A mesma ideia pode ser aplicada a colunas e regiões.

Figura 7 – Aplicação da técnica de par sozinho



(a) (b)

Fonte: Adaptada de Gordon e Longo (2006)

2.3.1.6 Par Oculto

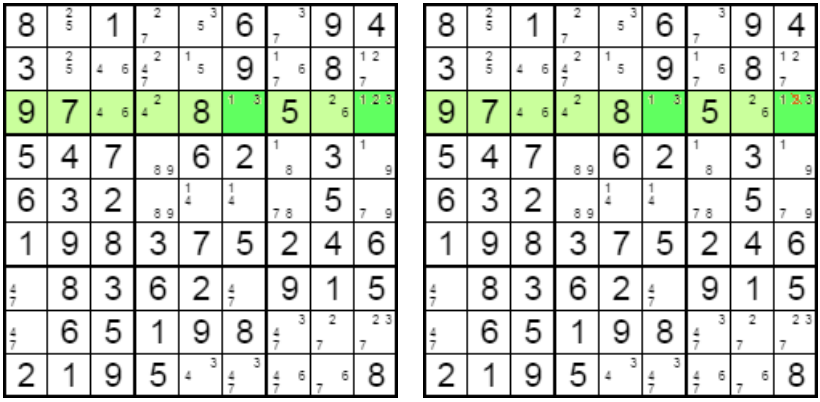
A técnica de *par oculto* é semelhante à técnica de *par sozinho*, mas as situações onde ela pode ser aplicada são um pouco mais difíceis de se identificar.

Se duas células vazias em uma linha, coluna ou região possuírem dois candidatos que não aparecem em nenhum outro lugar além destas células no mesmo grupo, estes candidatos devem ser inseridos nestas células (WILSON, 2005). Todos os candidatos restantes podem, portanto, ser removidos destas células. O mesmo raciocínio pode ser aplicado a trios de candidatos.

A Figura 8 apresenta um exemplo. Na linha destacada em (a), observa-se que existem apenas dois lugares onde os candidatos '1' e '3' podem ser inseridos. Eles poderiam ser vistos como dois pares sozinhos se uma das células não contivesse também o candidato '2'. Como '1' e '3' podem ser inseridos apenas nestas células, isto significa que o '2'

não é um candidato válido nestas células e pode ser removido, como destacado em (b).

Figura 8 – Aplicação da técnica de par oculto



(a) (b)

Fonte: Adaptada de Gordon e Longo (2006)

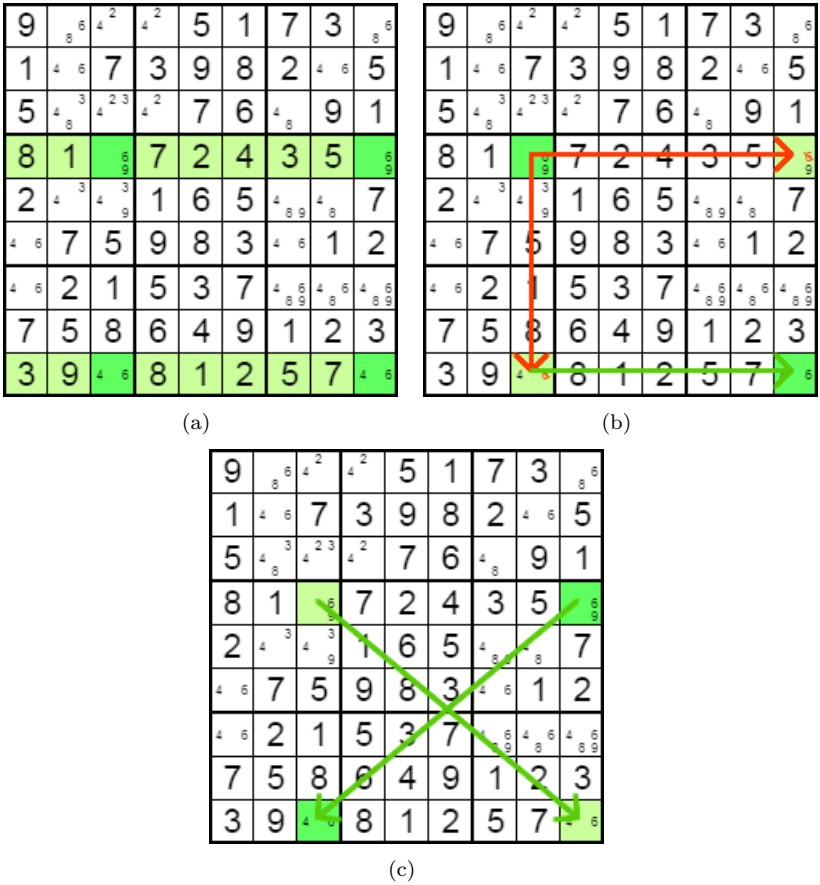
2.3.1.7 Cadeia Forçada

A *cadeia forçada* é uma técnica mais avançada e normalmente é utilizada como o último recurso de um jogador antes de partir para as técnicas de tentativa-erro.

Esta técnica se baseia no fato de que, em algumas ocasiões, a seleção de um valor para uma célula obrigará que outra célula seja preenchida com um valor específico para que as regras sejam satisfeitas (WILSON, 2005).

A Figura 9 mostra um exemplo, em uma situação conhecida como *Asa-X*. Observando os candidatos das células destacadas em (a), é possível perceber que se a célula superior esquerda for preenchida com o candidato '6', as células do mesmo grupo precisariam eliminar este candidato. Consequentemente, o único candidato restante para a célula inferior direita é o '6', como destacado em (b). Utilizando a mesma lógica, o preenchimento da célula superior direita com o candidato '6' força que a célula inferior esquerda também seja preenchida com '6', como mostrado em (c).

Figura 9 – Aplicação da técnica de cadeia forçada



(a) (b)

(c)

Fonte: Adaptada de Gordon e Longo (2006)

Existem diversas técnicas derivadas da Cadeia Forçada, como a *Asa-X* e a *Peixe-espada* (GORDON; LONGO, 2006). Estas técnicas nada mais são do que a aplicação da *cadeia forçada* em situações específicas, que podem ser observadas na grade com frequência. Gordon e Longo (2006) detalha algumas destas técnicas e as situações onde elas podem ser aplicadas.

2.3.2 Tentativa-erro

Alguns jogos não podem ser resolvidos apenas com as técnicas de eliminação de candidatos e, em muitos casos, é difícil identificar as situações onde estas técnicas podem ser aplicadas (WILSON, 2005). Nestes casos, o jogador é obrigado a supor um valor para uma célula e considerá-lo correto. A suposição pode, no entanto, se mostrar errônea posteriormente. Quando isso ocorre, o jogador deve voltar para o ponto onde fez a suposição, desfazendo todas as deduções seguintes, e fazer uma nova suposição para a célula.

A única técnica empregada nesta metodologia é selecionar a célula com o menor número de candidatos possíveis para fazer a suposição inicial. Esta escolha aumenta a probabilidade de selecionar o valor correto na suposição. Este método é chamado de *Nishio* (GORDON; LONGO, 2006).

2.4 ALGORITMOS PARA SOLUÇÃO

Jogos de Sudoku foram desenvolvidos originalmente para serem resolvidos por jogadores humanos, com lápis e papel. No entanto, é possível solucioná-los em praticamente tempo real com o auxílio de um computador e de um algoritmo bem projetado para realizar esta tarefa.

Esta seção apresenta algoritmos tipicamente utilizados para resolver sudokus. O foco principal é detalhar seu funcionamento. Contudo, existem outros aspectos que serão cobertos, como a dificuldade de implementação e o custo computacional. O objetivo é realizar uma análise inicial de algoritmos que poderiam ser empregados no sistema de controle do robô para resolver os jogos.

2.4.1 Sudoku Como Problema Computacional

O problema de resolver jogos de Sudoku é um tema de pesquisa nas áreas de ciência da computação e matemática (MCGUIRE; TUGEMANN; CIVARIO, 2012). Isto acontece pelo fato de que, quando generalizado para grades de dimensões $N^2 \times N^2$, ele se caracteriza como um problema *NP-completo*. Tal característica motivou diversas pesquisas sobre o jogo, que resultaram no desenvolvimento de diversos algoritmos que podem ser utilizados para solucioná-lo – apesar de nenhum, até o momento, fazê-lo em tempo polinomial e, enfim, responder a questão

$P = NP$ (GAREY; JOHNSON, 1979).

Mais especificamente, o Sudoku pode ser representado como um problema de *cobertura exata*, que é um dos 21 problemas NP-completos de Karp (MCGUIRE; TUGEMANN; CIVARIO, 2012). O problema de cobertura exata é um tipo de *problema de satisfação de restrições* (GAREY; JOHNSON, 1979). Assim como na maioria dos problemas desta classe, estabelecer a solução do Sudoku é um problema NP-completo.

Dada a grande quantidade de algoritmos publicados, e como a ampla maioria dos mesmos alcança a solução do problema em um tempo aceitável para a aplicação final, o presente trabalho se limitou a analisar os algoritmos mais utilizados no estado da arte em solucionadores determinísticos, a saber: *Backtracking*, *Dancing Links* e *Rule-based* (BERGGREN; NILSSON, 2012).

Além dos algoritmos apresentados a seguir, outros métodos relevantes merecem ser destacados. Ercsey-Ravasz e Toroczkai (2012) representa o Sudoku como um problema de satisfatibilidade booleana na forma normal conjuntiva e o resolve com um método determinístico de tempo contínuo. Esta solução é interessante pois não exige adivinhação e, consequentemente, elimina a necessidade de aplicar técnicas como o backtracking. Pacurib, Seno e Yusiong (2009) propõem uma solução baseada em colônia artificial de abelhas e *hill climbing*. Um solucionador que utiliza *Máquina de Boltzmann* é comparado com outros algoritmos por Berggren e Nilsson (2012). Por fim, Moon e Gunther (2006) consideram o Sudoku como um problema de satisfação de restrições e aplica técnicas de inteligência artificial para resolvê-lo.

2.4.2 Backtracking

O *backtracking* é um algoritmo de força-bruta refinado, empregado para resolver alguns tipos de problemas computacionais, em especial, problemas de satisfação de restrições. Este método enumera todas as combinações possíveis de resposta ao problema, que se apresentam como candidatas a solução do mesmo, e encontra a resposta correta por meio de uma busca em profundidade sobre as mesmas (GHEDIRA; DUBUISSON, 2013).

As combinações candidatas são representadas conceitualmente como nós de uma árvore. Cada candidata parcial é o pai de todas as candidatas que diferem da mesma por um único elemento combinatório. As folhas da árvore são as candidatas parciais que não podem mais aumentar. O algoritmo então percorre esta árvore, realizando uma busca

em profundidade. Em cada nó, o algoritmo verifica se a candidata parcial ainda pode se tornar uma solução válida para o problema. Se puder, o algoritmo verifica se esta candidata já é uma solução válida e, se este for o caso, a retorna como resposta. Após, continua a busca em todas as subárvores restantes, tentando encontrar outras soluções. Se a candidata não puder mais se tornar uma solução válida, todas as subárvores que possuem este nó como raiz são ignoradas. Esta verificação é realizada com base nas restrições do problema analisado.

Como o Sudoku pode ser representado como um problema de satisfação de restrições, é possível aplicar o *backtracking* para solucioná-lo (BERGGREN; NILSSON, 2012). Mesmo que este algoritmo execute em tempo exponencial, é plausível utilizá-lo, uma vez que não se conhece nenhum algoritmo que garanta uma resposta em tempo polinomial para solucionar problemas desta classe.

Considera-se que cada nó da árvore é um número em uma célula vazia. O algoritmo analisa as combinações candidatas preenchendo cada célula vazia da grade com um dígito, através de tentativas sucessivas. Quando todas as possibilidades são esgotadas para uma célula, ou seja, quando a inserção de qualquer número nela quebrar a restrição de unicidade de grupo, sabe-se que a combinação candidata não pode se tornar uma solução válida. Neste caso, o algoritmo retorna para última célula preenchida e substitui seu valor por um número diferente, efetuando o *backtracking*. Se toda a grade estiver preenchida e nenhuma restrição for violada, a solução foi encontrada. Este procedimento portanto realiza uma busca exaustiva da solução e certamente a encontrará ao fim do tempo de execução se ela existir. Na verdade, este algoritmo nada mais é do que a aplicação da metodologia de tentativa-erro na forma de um algoritmo computacional, descrita na seção anterior.

O método mais trivial de executar o *backtracking* em um problema de Sudoku é tomar a primeira célula vazia encontrada na grade e preenchê-la sequencialmente com os números de ‘1’ até ‘9’, como mostra a Listagem 1, em forma de pseudo-código.

Listagem 1: BACKTRACKING aplicado ao problema do Sudoku

Input: Uma matriz de dimensões 9×9
$M = \{\{a_{11}, \ldots, a_{19}\}, \ldots, \{a_{91}, \ldots, a_{99}\}\}$ de inteiros, representando a grade de um jogo Sudoku parcialmente preenchida

Output: Uma matriz de dimensões 9×9
$S = \{\{a_{11}, \ldots, a_{19}\}, \ldots, \{a_{91}, \ldots, a_{99}\}\}$ de inteiros, representando a grade do jogo descrito por M resolvido

```
(x, y) ← encontrar uma célula vazia de M
for c in candidatos de M em (x, y) do
    M[x][y] ← c
    S ← Backtracking(M)
    if S é válido e está totalmente preenchido then
        return S
return nenhuma solução encontrada
```

Neste algoritmo, a grade de Sudoku é representada por uma matriz de inteiros com dimensões 9×9. As células vazias são representadas pelo valor zero. Existe uma série de variações interessantes do algoritmo que o tornam mais eficiente para resolver problemas de Sudoku. Segundo Ghedira e Dubuisson (2013), as mais relevantes são a utilização da heurística de *Minimum Remaining Values* (MRV) e a implementação de *Forward Checking* (FC).

O MRV é um método heurístico que pode ser empregado para seleção da próxima célula vazia a ser analisada pelo algoritmo na resolução do Sudoku (GHEDIRA; DUBUISSON, 2013). A ideia básica deste método é que a escolha aleatória de um valor para a variável com o menor número de candidatos possui maior probabilidade de sucesso e, consequentemente, menores chances de se mostrar falha futuramente.

O MRV pode ser compreendido como o método de *Nishio*, apresentado na seção anterior. A implementação deste método exige que uma lista de candidatos seja mantida para cada célula da grade, e que estas listas sejam atualizadas todas as vezes que uma célula no mesmo grupo tenha seu valor modificado.

O FC, por sua vez, consiste em verificar, após cada atribuição de valor a uma variável, todas as restrições influenciadas por esta variável (GHEDIRA; DUBUISSON, 2013). Ele é útil porque reduz o domínio de

variáveis livres que aparecem nestas restrições. Se o domínio de alguma variável livre se reduzir ao conjunto vazio, então um *backtrack* é realizado.

Para aplicá-lo no Sudoku, faz-se uso da mesma lista de candidatos utilizada na implementação MRV. Após a modificação do valor de uma célula, basta verificar se as listas de candidatos das demais células do mesmo grupo estão vazias. Se este for o caso para pelo menos uma célula, um *backtrack* é executado. O benefício do uso desta técnica é a capacidade de identificar com antecedência que um valor incorreto foi atribuído a alguma célula.

A Listagem 2 descreve, em pseudo-código, o *backtracking* apresentado anteriormente na Listagem 1 aperfeiçoado com as técnicas de MRV e FC.

Listagem 2: BACKTRACKING utilizando Minimum Remaining Values e Forward Checking aplicado ao problema do Sudoku

Input: Uma matriz de dimensões 9×9 $M = \{\{a_{11}, \ldots, a_{19}\}, \ldots, \{a_{91}, \ldots, a_{99}\}\}$ de inteiros, representando a grade de um jogo Sudoku parcialmente preenchida

Output: Uma matriz de dimensões 9×9 $S = \{\{a_{11}, \ldots, a_{19}\}, \ldots, \{a_{91}, \ldots, a_{99}\}\}$ de inteiros, representando a grade do jogo descrito por M resolvido

```
// Minimum Remaining Values
(x, y) ← encontrar uma célula vazia de M com o menor número de candidatos
for c in candidatos de M em (x, y) do
    v ← false
    M[x][y] ← c
    for o in células da mesma linha, coluna e região que (x, y) de M do
        Atualizar lista de candidatos de o
        if não existem candidatos para o then
            // Forward checking
            v ← true
            break
    if v then
        continue
    S ← Backtracking(M)
    if S é válido e está totalmente preenchido then
        return S
return nenhuma solução encontrada
```

Uma série de outras melhorias podem ser realizadas no algoritmo básico de *backtracking*. (GHEDIRA; DUBUISSON, 2013) apresenta uma série de possibilidades.

2.4.3 Dancing Links

Dancing Links é um algoritmo proposto por Knuth (2000) que encontra soluções para problemas de cobertura exata, categoria de problemas a qual o Sudoku pertence. Antes de apresentar este algoritmo, é necessário definir formalmente o que é a cobertura exata e o que é o problema de cobertura exata.

Seja $\mathcal{S}$ uma família de subconjuntos de um conjunto finito $\mathcal{X}$. Uma cobertura exata de $\mathcal{X}$ por $\mathcal{S}$ é uma partição de $\mathcal{X}$, onde cada elemento pertencente a $\mathcal{X}$ está contido em $\mathcal{S}$ exatamente uma vez (GAREY; JOHNSON, 1979). Por exemplo, se $\mathcal{X} = \{x_1, x_2, x_3\}$ e $\mathcal{S} = \{\{x_1\}, \{x_1, x_2\}, \{x_2, x_3\}\}$, então $S^* = \{\{x_1\}, \{x_2, x_3\}\}$ é uma cobertura exata de $\mathcal{X}$ por $\mathcal{S}$.

O *problema da cobertura exata* é um problema de decisão para determinar se uma cobertura exata existe(GAREY; JOHNSON, 1979). Ele é um problema NP-completo e é um dos 21 problemas NP-completos de Karp (MCGUIRE; TUGEMANN; CIVARIO, 2012). Normalmente, um problema de cobertura exata é representado por uma matriz de incidência ou por um grafo bipartido (GHEDIRA; DUBUISSON, 2013).

A representação em matriz utiliza uma linha para cada subconjunto em $\mathcal{S}$ e uma coluna para cada elemento em X (GHEDIRA; DUBUISSON, 2013). O valor de uma interseção linha/coluna em particular é preenchida com '1' se o subconjunto correspondente da linha contiver o elemento correspondente da coluna, sendo definido como '0' caso contrário. Nesta representação, uma cobertura exata é dada a seleção de um subconjunto de linhas onde cada coluna contém o '1' em uma e somente uma linha. Seguindo o exemplo anterior, a sua representação em forma de matriz de incidência é dada pela Equação (2.1).

$$\begin{bmatrix} & x_1 & x_2 & x_3 \\ \{x_1\} & 1 & 0 & 0 \\ \{x_1, x_2\} & 1 & 1 & 0 \\ \{x_2, x_3\} & 0 & 1 & 1 \end{bmatrix} \tag{2.1}$$

O subconjunto $S^* = \{\{x_1\}, \{x_2, x_3\}\}$ é uma cobertura exata pois cada elemento está contido em exatamente um subconjunto selecionado, ou seja, cada coluna contém o número '1' em exatamente uma linha, como mostra a matriz de incidência.

O Dancing Links recebe como entrada um problema de cobertura exata representado na forma de uma matriz de incidência, tal como a do exemplo. A ideia básica é selecionar, a cada passo, uma linha

da matriz. A seleção remove da matriz esta linha e as colunas onde a interseção com a mesma são '1'. Além disso, outras linhas que possuem o valor 1 para alguma destas colunas também são removidas, uma vez que seu objetivo é selecionar um subconjunto de linhas tal que o '1' apareça em cada coluna exatamente uma vez. Este processo é repetido sucessivamente, e seu resultado é uma matriz reduzida a cada repetição (KNUTH, 2000).

As linhas selecionadas em cada passo são adicionadas a uma lista, que representa a solução parcial. Se a matriz reduzida não possuir mais colunas, então o subconjunto de linhas selecionadas representa uma solução. Se por outro lado a matriz reduzida ainda possuir colunas, mas não existir mais nenhuma linha para ser selecionada, a seleção atual não representa uma solução. O algoritmo verifica todas as combinações possíveis, eliminando antecipadamente resultados falhos de forma semelhante ao *Backtracking*, e encontra todas as soluções possíveis para o problema se elas existirem.

O *Dancing Links* é apresentando na forma de pseudo-código na Listagem 3.

Listagem 3: DANCING LINKS

Input: Uma matriz de incidência A onde as colunas representam o conjunto de elementos X e linhas representam a coleção de subconjuntos $\mathcal{S}$ de X; Uma solução parcial S;
Output: Uma lista de linhas S de M, representando a subcoleção $\mathcal{S}^*$, cobertura exata da entrada (se existir)

```
1  c ← uma coluna de A
2  for l in linhas de A tal que A[r][c] = 1 do
3      Adicionar l em S
4      for j in colunas de A tal que A[l][j] = 1 do
5          for i in linhas de A tal que A[i][j] = 1 do
6              Remover i de A
7          Remover j de A
8      S ← DancingLinks(A, S);
9      if A não possui nenhuma coluna then
10         return S
11     for j in colunas removidas de A do
12         Restaurar j em A
13     for i in linhas removidas de A do
14         Restaurar i em A
15 return nenhuma solução encontrada
```

Este algoritmo é normalmente implementado com listas duplamente encadeadas (KNUTH, 2000). Desta forma, as operações de remoção e restauração de linhas e colunas na matriz podem assim ser realizadas com apenas duas operações de atribuição. Knuth (2000) observou que esta abordagem é consideravelmente mais eficiente do que realizar verificações sobre a matriz completa. Isto acontece porque quando uma linha é selecionada, uma coluna inteira precisa ser verificada em busca do número '1'. Após a seleção da linha, esta precisa ser novamente percorrida e as colunas onde o valor da interseção é '1' precisam ser verificadas por completo. A simples utilização de listas encadeadas, implementando uma matriz esparsa[1] onde somente as células contendo '1' são mantidas, diminui a complexidade destas buscas de O(n) para

[1] Uma matriz esparsa é uma matriz que possui uma grande quantidade de elementos vazios, ou iguais a zero.

O(1).

Para aplicar o algoritmo na solução do Sudoku, é necessário representar os jogos na forma de uma matriz de incidência. Existem quatro restrições básicas que a solução final do jogo deve ser satisfazer:

1. **Restrição de interseção:** Cada célula da grade deve conter um e somente um número;

2. **Restrição de linha:** Cada linha deve conter uma e somente uma ocorrência de cada número;

3. **Restrição de coluna:** Cada coluna deve conter uma e somente uma ocorrência de cada número;

4. **Restrição de região:** Cada região deve conter uma e somente uma ocorrência de cada número;

A primeira restrição, apesar de parecer óbvia, é necessária para garantir que exista apenas um número por célula, ou seja, os subconjuntos de $\mathcal{S}^*$ contenham apenas um elemento cada.

Seguindo estas restrições, matriz de incidência deve especificar, em suas linhas, todas as possibilidades existentes. Como no Sudoku existem 9 linhas, 9 colunas e 9 candidatos para cada célula, a quantidade de linhas na matriz deve ser $9 \times 9 \times 9 = 729$. Com relação às colunas, são utilizadas $9 \times 9 = 81$ para restrições de combinação linha-coluna ($L_M C_N$), $9 \times 9 = 81$ para restrições de combinação linha-número ($L_M \#_N$), $9 \times 9 = 81$ para restrições de combinação coluna-número ($C_M \#_N$) e $9 \times 9 = 81$ restrições para a combinação região-número ($R_M C_N$), totalizando 324 colunas.

Desta forma, um jogo de Sudoku pode ser representado por uma matriz de incidência de dimensões 729×324. As matrizes abaixo ilustram a configuração geral da mesma. Apesar de ser difícil representar a matriz inteira, é possível ilustrar a configuração geral da mesma por meio das submatrizes apresentadas pelas Equações 2.2, 2.3, 2.4 e 2.5.

$$\begin{bmatrix} & L1C1 & L1C2 & \cdots \\ L1C1\#1 & 1 & 0 & \cdots \\ L1C1\#2 & 1 & 0 & \cdots \\ L1C1\#3 & 1 & 0 & \cdots \\ \vdots & \vdots & \vdots & \ddots \\ L1C1\#9 & 1 & 0 & \cdots \\ L1C2\#1 & 0 & 1 & \cdots \\ L1C2\#2 & 0 & 1 & \cdots \\ L1C2\#3 & 0 & 1 & \cdots \\ \vdots & \vdots & \vdots & \ddots \\ L1C2\#9 & 0 & 1 & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix} \tag{2.2}$$

$$\begin{bmatrix} & L1\#1 & L1\#2 & \cdots \\ L1C1\#1 & 1 & 0 & \cdots \\ L1C1\#2 & 0 & 1 & \cdots \\ \vdots & \vdots & \vdots & \ddots \\ L1C1\#9 & 0 & 0 & \cdots \\ L1C2\#1 & 1 & 0 & \cdots \\ L1C2\#2 & 0 & 1 & \cdots \\ \vdots & \vdots & \vdots & \ddots \\ L1C2\#9 & 0 & 0 & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix} \tag{2.3}$$

$$\begin{bmatrix} & C1\#1 & C1\#2 & \cdots \\ L1C1\#1 & 1 & 0 & \cdots \\ L1C1\#2 & 0 & 1 & \cdots \\ \vdots & \vdots & \vdots & \ddots \\ L1C1\#9 & 0 & 0 & \cdots \\ \vdots & \vdots & \vdots & \ddots \\ L2C1\#1 & 1 & 0 & \cdots \\ L2C1\#2 & 0 & 1 & \cdots \\ \vdots & \vdots & \vdots & \ddots \\ L2C1\#9 & 0 & 0 & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix} \tag{2.4}$$

$$
\begin{bmatrix}
 & R1\#1 & R1\#2 & \cdots \\
L1C1\#1 & 1 & 0 & \cdots \\
L1C1\#2 & 0 & 1 & \cdots \\
\vdots & \vdots & \vdots & \ddots \\
L1C2\#1 & 1 & 0 & \cdots \\
L1C2\#2 & 0 & 1 & \cdots \\
\vdots & \vdots & \vdots & \ddots \\
L1C3\#1 & 1 & 0 & \cdots \\
L1C3\#2 & 0 & 1 & \cdots \\
\vdots & \vdots & \vdots & \ddots \\
L1C4\#1 & 0 & 0 & \cdots \\
L1C4\#2 & 0 & 0 & \cdots \\
\vdots & \vdots & \vdots & \ddots \\
L2C1\#1 & 1 & 0 & \cdots \\
L2C1\#2 & 0 & 1 & \cdots \\
\vdots & \vdots & \vdots & \ddots \\
L2C2\#1 & 1 & 0 & \cdots \\
L2C2\#2 & 0 & 1 & \cdots \\
\vdots & \vdots & \vdots & \ddots
\end{bmatrix}
\tag{2.5}
$$

Após montar a matriz de incidência para o jogo a ser resolvido, basta utilizá-la como parâmetro de entrada para o *Dancing Links* – a saída retornará a solução do jogo, se ela existir.

2.4.4 Rule-based

O *Rule-based* é um algoritmo solucionador de Sudoku heurístico. Ele utiliza os métodos de solução manual, como os apresentados na seção anterior, para resolver os jogos (BERGGREN; NILSSON, 2012). Como o nome sugere, este algoritmo se baseia nas regras do Sudoku para inferir logicamente o número que deve ser inserido em uma célula vazia ou eliminar candidatos que são impossíveis na mesma. Ele resolve jogos de Sudoku como um humano faria, mas com grande velocidade e sendo capaz de identificar todas as situações em que os métodos implementados podem se aplicar.

Nesta abordagem, um conjunto de métodos de solução manual para o problema do Sudoku é implementado, construindo uma base

de conhecimento. O algoritmo itera sobre estes métodos, verificando se eles podem ser aplicados ao jogo e executando-os quando possível. Quando uma célula vazia é preenchida ou candidatos são eliminados, ou seja, o estado do jogo é alterado, o algoritmo retorna ao primeiro método e continua a iteração. Este procedimento se repete até que a solução do jogo seja alcançada

O algoritmo prioriza a utilização de regras baseadas na metodologia de eliminação de candidatos, deixando as técnicas de tentativa-erro como últimas alternativas. É uma boa prática ainda ordenar as regras de acordo com sua complexidade ou probabilidade de ocorrência durante as iterações: prioriza-se a aplicação de regras que exigem menor custo computacional para verificação e execução, deixando para executar as regras mais complexas somente se necessário. Espera-se, com isso, minimizar o tempo necessário para obter a solução final.

Segundo Berggren e Nilsson (2012), quanto mais regras forem implementadas, e quanto maior o refinamento das mesmas, maior a tendência de que o algoritmo desempenhe de forma eficiente. A utilização de muitas regras na base de conhecimento, no entanto, pode provocar atrasos devido a quantidade de verificações realizadas. Em especial para situações onde nenhuma conclusão lógica pode ser tomada e a metodologia de tentativa-erro é a única saída, este atraso pode ser considerável. Este cenário é frequente em problemas de grande dificuldade.

A Listagem 4 apresenta o algoritmo *Rule-based* com os métodos apresentados na seção 2.3 implementados, a saber:

1. **Candidato Único:** se as possibilidades de uma célula vazia se reduzem a um único candidato, então este candidato é deve ser inserido na célula (GORDON; LONGO, 2006);

2. **Posição Única:** se existe apenas uma célula vazia na linha, coluna ou região onde um número é candidato, este número deve ser inserido na célula (GORDON; LONGO, 2006);

3. **Pares Sozinhos:** se os mesmos números são candidatos em duas células distintas na mesma linha, coluna ou região, estes números devem ser removidos da lista de candidatos das demais células no mesmo grupo (linha, coluna ou região). O mesmo método pode ser utilizado com três e quatro candidatos/células (GORDON; LONGO, 2006);

4. **Pares Ocultos:** se duas células vazias em uma linha, coluna ou região possuírem dois candidatos que não aparecem em ne-

nhum outro lugar além destas células no mesmo grupo, estes candidatos devem ser inseridos nestas células. Todos os candidatos restantes podem, portanto, ser removidos destas células. O mesmo raciocínio pode ser aplicado a trios de candidatos (GORDON; LONGO, 2006);

5. **Linha Candidata:** se todas as células vazias onde um número em particular é candidato estão na mesma linha (ou coluna) e na mesma região, então este candidato pode ser eliminado em outras células vazias da mesma linha (ou coluna) fora desta região (GORDON; LONGO, 2006);

6. **Pares Duplos:** se dois pares de células em regiões distintas possuem um valor como candidato na mesma linha (ou coluna), este valor pode ser removido da última região na mesma linha (ou coluna) (GORDON; LONGO, 2006);

7. **Asa-X (Cadeia Forçada):** se existem apenas duas células vazias onde um valor é candidato em cada uma de duas linhas diferentes, e estas células estão nas mesmas colunas, então este valor pode ser removido da lista de candidatos de todas outras células nestas colunas (GORDON; LONGO, 2006);

8. **Nishio:** técnica de tentativa-erro onde as células com o menor número de candidatos são preenchidas primeiro. Implementada pelo *backtracking* com heurística de MRV (GORDON; LONGO, 2006);

Todos os métodos descritos, com exceção de Nishio, podem ser implementados de forma a executar em tempo polinomial. Desta forma, o algoritmo produzirá uma solução polinomial caso a última regra não precise ser aplicada para obter uma solução.

Listagem 4: RULE-BASED aplicado ao Sudoku

Input: Uma matriz de dimensões 9×9 $M = \{\{a_{11}, \ldots, a_{19}\}, \ldots, \{a_{91}, \ldots, a_{99}\}\}$ de inteiros, representando a grade de um jogo Sudoku parcialmente preenchida

Output: Uma matriz de dimensões 9×9 $S = \{\{a_{11}, \ldots, a_{19}\}, \ldots, \{a_{91}, \ldots, a_{99}\}\}$ de inteiros, representando a grade do jogo descrito por M resolvido

```
1  while true do
2      if aplicarCandidatoUnico(M) then
3          if M é válido e está totalmente preenchido then
4              return M
5          continue
       // Aplicar os demais métodos...
6      if aplicarAsaX(M) then
7          if M é válido e está totalmente preenchido then
8              return M
9          continue
10     break
   // Backtracking com MRV e FC
11 (x, y) ← encontrar uma célula vazia de M com o menor
   número de candidatos
12 for c in candidatos de M em (x, y) do
13     v ← false
14     M[x][y] ← c
15     for o in células da mesma linha, coluna e região que
       (x, y) de M do
16         Atualizar lista de candidatos de o
17         if não existem candidatos para o then
18             v ← true
19             break
20     if v then
21         continue
22     S ← RuleBased(M)
23     if S é válido e está totalmente preenchido then
24         return S
25 return nenhuma solução encontrada
```

3 AQUISIÇÃO E PROCESSAMENTO DE IMAGENS

O campo do processamento digital de imagens se refere ao processamento de imagens digitais por um computador digital (SOLOMON; BRECKON, 2010). Segundo Gonzalez e Woods (2008), o processamento digital de imagens envolve processos cujas entradas e saídas são imagens e, além disso, envolve processos de extração de atributos de imagens até – e inclusive – o reconhecimento de objetos individuais. O interesse nos métodos de processamento de imagens provém de duas áreas principais de aplicação: melhora das informações visuais para a interpretação humana e processamento de dados de imagens para armazenamento, transmissão e representação, considerando a percepção automática por máquinas (GONZALEZ; WOODS; EDDINS, 2003).

Este capítulo apresenta as bases teóricas necessárias para desenvolver o sistema de visão de máquina utilizado para reconhecer jogos de Sudoku, parte integrante do sistema de controle do robô proposto. Inicialmente, são apresentados conceitos básicos sobre processamento de imagens. Após, técnicas de processamento empregadas no sistema, que incluem a detecção de linhas e formas, são detalhadas. Finalmente, métodos de reconhecimento óptico de caracteres (OCR) utilizados para reconhecer as pistas são discutidos.

3.1 IMAGEM DIGITAL

Gonzalez e Woods (2008) define uma imagem como uma função bidimensional, $f(x, y)$, em que x e y são coordenadas de um plano e a amplitude de f em qualquer par de coordenadas (x, y) é chamada de *intensidade* da imagem nesse ponto. Quando x, y e os valores de intensidade de f são quantidades finitas e discretas, a imagem é dita uma *imagem digital*. As imagens digitais são uma forma conveniente de representação de imagens contínuas (GONZALEZ; WOODS, 2008).

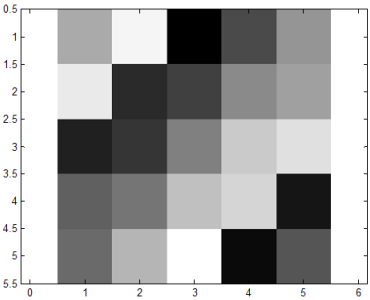
As cenas do mundo real, percebidas pelos olhos humanos, são imagens contínuas (SONKA; HLAVAC; BOYLE, 2014). A importância da representação de imagens na forma digital está diretamente relacionada com o desenvolvimento dos computadores digitais (SOLOMON; BRECKON, 2010). Os computadores e outros dispositivos digitais operam em passos discretos e armazenam dados na forma de bits discretos, impossibilitando o processamento e o armazenamento de sinais contínuos. Imagens digitais, por outro lado, podem ser facilmente armazenadas,

processadas e transmitidas pelos mesmos. Esta característica torna a representação digital de imagens conveniente na solução de diversos problemas, pois reduz a complexidade para realizar processamentos e, consequentemente, diminui os custos nestes processos.

Imagens digitais são compostas de um número finito de elementos, chamados de *pixels*, cada um com localização e valor de intensidade específicos (GONZALEZ; WOODS; EDDINS, 2003). Elas são normalmente representadas por uma matriz de 2 dimensões, contendo M linhas e N colunas, sendo (x, y) coordenadas discretas para células desta matriz. Para fins de praticidade e clareza na notação, utilizam-se números inteiros para estas coordenadas – $x = 0, 1, 2, ..., M-1$ e $y = 0, 1, 2, ..., N-1$. Desta forma, por exemplo, o valor da imagem digital na origem é $f(0, 0)$, e o próximo valor de coordenada ao longo da primeira linha é $f(0, 1)$. De um modo geral, a imagem digital é uma matriz de pontos, onde cada célula da matriz possui um valor de intensidade proporcional à quantidade de iluminação da fonte que incide (*iluminação*) e é refletida (refletância) na cena (GONZALEZ; WOODS, 2008). A iluminação e a refletância são responsáveis, portanto, por caracterizar as cores observadas em uma imagem (SONKA; HLAVAC; BOYLE, 2014).

A Figura 10 apresenta uma imagem de 5×5 *pixels* ampliada, ilustrando o conceito de imagem digital. Nesta imagem, o nível de cinza de cada ponto é proporcional ao valor da intensidade de f neste ponto. Este é um tipo de representação visual de imagens digitais.

Figura 10 – Representação visual em escala de cinza de uma imagem digital



Fonte: Elaborada pelo autor

Além da representação em níveis de cinza, outra representação conveniente de imagens é realizada por meio de matrizes numéricas (GONZALEZ; WOODS; EDDINS, 2003). Esta representação é comumente

utilizadas para representar quantitativamente imagens digitais para processamento, onde cada posição da matriz representa a localização de um *pixel*, e o valor numérico a sua intensidade. A matriz apresentada pela Equação (3.1) ilustra este conceito, representando a imagem da Figura 10 por meio de uma matriz de números.

$$f(x,y) = \begin{bmatrix} 170 & 245 & 0 & 74 & 148 \\ 235 & 42 & 64 & 138 & 160 \\ 32 & 54 & 128 & 150 & 225 \\ 96 & 118 & 192 & 213 & 21 \\ 106 & 181 & 255 & 10 & 85 \end{bmatrix} \tag{3.1}$$

A Equação (3.2) estende algebricamente o conceito para imagens digitais de tamanho $M \times N$. Os dois lados desta equação são formas equivalentes de representar quantitativamente uma imagem digital. O lado direito desta matriz é uma matriz de números reais, e cada elemento da mesma é um *pixel*.

$$f(x,y) = \begin{bmatrix} f(0,0) & f(0,1) & \cdots & f(0,N-1) \\ f(1,0) & f(1,1) & \cdots & f(1,N-1) \\ \vdots & \vdots & \ddots & \vdots \\ f(M-1,0) & f(M-1,1) & \cdots & f(M-1,N-1) \end{bmatrix} \tag{3.2}$$

É importante observar que, por convenção, a origem de uma imagem digital se situa no pixel superior esquerdo ($f(0,0)$), com o eixo x positivo se estendendo para direita e o eixo y positivo se estendendo para baixo. Segundo Gonzalez e Woods (2008), essa representação é utilizada por fatores históricos. Os primeiros dispositivos de visualização de imagens varrem uma imagem começando do canto superior esquerdo e se movendo para a direita, uma linha por vez.

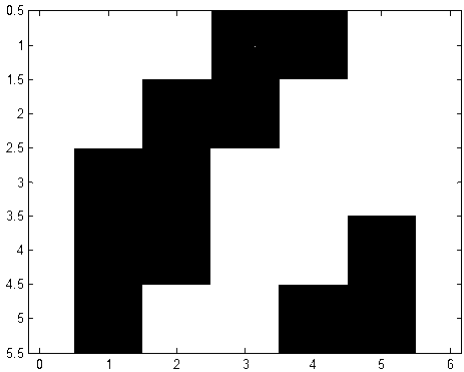
3.2 TIPOS DE IMAGENS DIGITAIS

Sonka, Hlavac e Boyle (2014) classifica as imagens digitais de acordo com os valores de intensidade dos seus pixels em três tipos: *binárias*, *em escala de cinza* e *coloridas*. Esses tipos são apresentados a seguir.

3.2.1 Imagens Binárias

Uma imagem binária é uma imagem que possui apenas dois valores possíveis de intensidade para cada pixel (GONZALEZ; WOODS, 2008). Tipicamente, os dois valores de intensidade utilizados na representação de uma imagem binária são '0' e '1', representando as cores *preto* (*background*) e *branco* (*foreground*), respectivamente. A Figura 11 apresenta uma imagem binária.

Figura 11 – Exemplo de imagem binária



Fonte: Elaborada pelo autor

A representação binária de imagens possui grande utilidade no processamento digital de imagens, sendo a representação utilizada para executar diversas operações (GONZALEZ; WOODS; EDDINS, 2003). São exemplos de operações que utilizam imagens binárias as operações morfológicas e a segmentação.

3.2.2 Imagens em Escala de Cinza

Imagens em escala de cinza são imagens onde o valor de cada pixel representa apenas a informação de intensidade (GONZALEZ; WOODS, 2008). Imagens deste tipo são compostas exclusivamente por tons de cinza, variando de preto (na menor intensidade) a branco (na maior intensidade). A Figura 10, apresentada anteriormente, é uma imagem em escala de cinza.

A intensidade de um pixel é expressa numericamente por um

intervalo fechado entre um valor mínimo e um valor máximo, que definem *completamente preto* e *completamente branco*. Imagens em escala de cinza são normalmente representadas em formato digital utilizando 8 bits por pixel, o que permite que 256 intensidades diferentes incluindo preto e branco – daí os valores de 0 até 255 utilizados na matriz numérica da Equação (3.1). Em algumas aplicações onde um alto nível de detalhamento das imagens é exigido, como na área médica, 16 bits por pixel podem ser utilizados. Quanto mais bits utilizados na representação de um pixel em uma imagem digital, mais níveis de cinza podem ser representados e, portanto, mais rica em detalhes será a imagem. Gonzalez, Woods e Eddins (2003) esclarece que isso se deve ao fato de que o aumento do número de bits por pixel diminui o intervalo entre valores no processo de quantização da imagem contínua.

Imagens em escala de cinza podem ser transformadas em imagens binárias por meio de uma operação chamada *thresholding* (SOLOMON; BRECKON, 2010). No processo clássico de thresholding, cada pixel em uma imagem é substituído por um pixel de fundo se a intensidade é menor que uma constante K, ou por um pixel branco caso contrário. A imagem apresentada na Figura 11, por exemplo, foi obtida realizando thresholding da imagem na Figura 10. A operação de *thresholding* é apresentada com detalhes na seção 3.4.5.

3.2.3 Imagens Coloridas

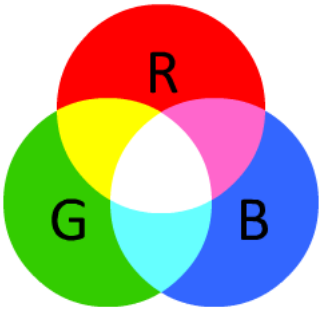
Em imagens coloridas, os pixels passam a ser representados por um sistema de cores, ao invés de um único valor numérico de intensidade (GONZALEZ; WOODS, 2008). Segundo Sonka, Hlavac e Boyle (2014) definem sistemas de cores como tentativas de organizar informações sobre a percepção cromática humana, criando um modelo matemático para representação de cores. Na natureza, são encontrados dois sistemas cromáticos: os sistemas *Aditivos* e *Subtrativos*.

Os sistemas aditivos usam o mesmo princípio que o sistema de visão humano. As cores são formadas por três componentes de cor, correspondentes à três frequências, que são adicionados em diferentes proporções de intensidade ao raio luminoso para produzir as diferentes cores. Sistemas aditivos descrevem muito bem o comportamento da cor em raios de luz, sendo amplamente utilizados em monitores de vídeo, televisão, câmeras e dispositivo que tenham que gerar ou detectar cores. A discussão sobre sistemas de cores neste trabalho será restriginda aos sistemas aditivos.

Segundo (GONZALEZ; WOODS; EDDINS, 2003), o sistema *RGB* é o mais popular sistema de cores utilizado no mundo. Ele é um sistema aditivivo que descreve a cor como uma composição de 3 cores (frequências) primárias: o vermelho (***R****ed*), o verde (***G***reen) e o azul (***B***lue). Todas as outras cores, são formadas adicionando estas cores em diferentes intensidades. Então, os valores dos componentes (canais) RGB de uma cor representam a intensidade para aquele componente na composição da cor. O valor zero indica intensidade zero, ou seja, nenhuma luz. Os três componentes no valor mínimo representam o preto, que é a ausência total de luz, mas se forem levados ao máximo, a cor obtida será o branco, que é a composição das três frequências (cores) primárias em intensidade máxima.

Na representação computacional mais comum de imagens digitais RGB, os valores de cada componente variam de 0 à 255, com cada componente sendo representado por 8 bits. Desta forma, os valores $RGB(0, 0, 0)$ representam o preto e os valores $RGB(255, 255, 255)$ representam o branco. Juntos, os três valores (24 bits) produzem $256 \times 256 \times 256$ combinações, que correspondem a 16.777.216 cores. Este método de representação, onde cada pixel é representado por 24 ou mais bits (8 bits por canal), é chamado de *True Color* (SONKA; HLAVAC; BOYLE, 2014). Acredita-se que o olho humano consegue distinguir algo em torno de 10 milhões de cores (GONZALEZ; WOODS, 2008). O sistema se chama *true color* justamente por mostrar mais cores que o olho humano pode ver e, consequentemente, dá a ilusão de cores reais. A Figura 12 mostra como as cores RGB são combinadas para formar as outras cores.

Figura 12 – Sistema RGB



Fonte: Adaptado de Burger e Burge (2008)

Qualquer cor representada no sistema RGB pode ser convertida em seu nível aproximado de cinza (intensidade). Para tanto, realiza-se uma média ponderada dos componentes, com pesos de 30% para o vermelho, 59% para o verde e 11% para o azul, independentemente da escala utilizada para os canais (BURGER; BURGE, 2008). O nível resultante é o valor de intensidade em cinza equivalente. Os valores dos pesos utilizados estão relacionados com a sensibilidade visual do olho humano para as cores primárias. A Equação (3.3) descreve a operação.

$$I_{cinza} = 0.299 \times I_{vermelho} + 0.587 \times I_{verde} + 0.114 \times I_{azul} \qquad (3.3)$$

3.3 AQUISIÇÃO DE IMAGENS

O processo de obter imagens de uma fonte, de forma a transformá-las em informação útil para armazenamento, reprodução e processamento, é chamado de *aquisição de imagens*. A aquisição é a primeira etapa de um sistema de processamento de imagens, uma vez que sem uma imagem nenhum processamento pode ser realizado (GONZALEZ; WOODS, 2008).

Conforme Davies (2012), um dos principais objetivos da aquisição é possuir uma fonte de entrada que opere de forma tão controlada que a mesma imagem possa ser perfeitamente reproduzida se as condições de captura forem as mesmas, minimizando a presença e facilitando a remoção de anomalias. Este processo é de extrema importância para o funcionamento dos sistemas de processamento de imagens, visto que se uma imagem não for obtida em condições minimamente satisfatórias, o sistema pode não ser capaz de executar com sucesso as tarefas para as quais foi projetado, mesmo se técnicas de melhoramento forem aplicadas posteriormente (CORKE, 2011).

As imagens digitais são tipicamente obtidas por meio de dispositivos ópticos (*hardware*), que capturam imagens analógicas (cenas reais) e as convertem para representações digitais. São exemplos dispositivos de aquisição as câmeras fotográficas, *webcams*, *scanners*, telescópios e até mesmo sensores de presença de luz.

3.3.1 Amostragem e Quantização

As imagens fotográficas são obtidas do mundo real através de câmeras ou sensores que captam luz. A imagem capturada em um filme

fotográfico representa bem a imagem real. Gonzalez e Woods (2008) explicam que o filme define um plano limitado por um retângulo, onde cada posição nesse plano contém a informação de cor relativa aquela posição, ou seja, a imagem neste caso é um sinal contínuo de cor 2D, onde o domínio é o plano e o contradomínio é o espaço de cor.

Os computadores digitais, no entanto, trabalham apenas com valores discretos. Para que uma imagem possa ser processada, armazenada e transmitida por um computador, é necessário portanto convertê-la para o tempo discreto antes. Para gerar uma imagem digital, $f(x, y)$ deve ser digitalizada ao longo de x e y e da amplitude $z = f(x, y)$. Esta tarefa é realizada por meio dos processos de *amostragem* e *quantização*, respectivamente.

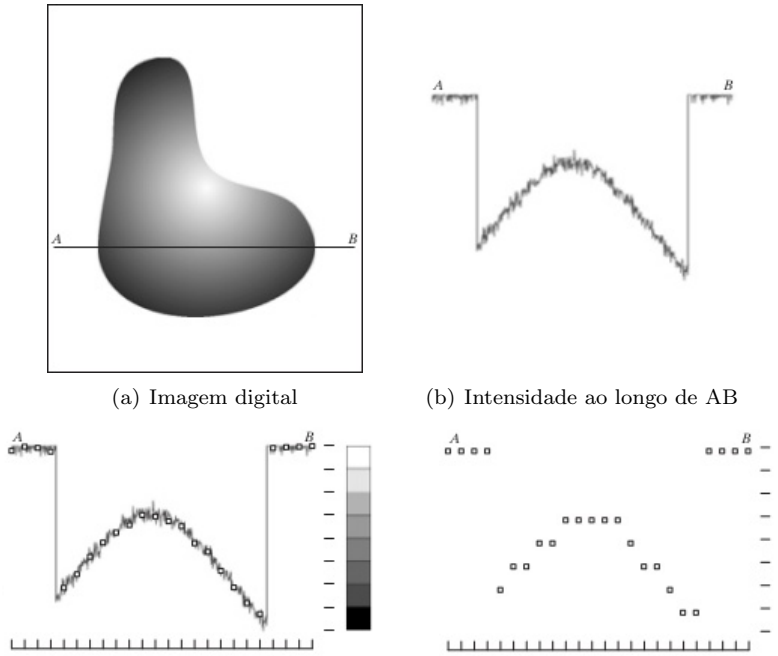
Na amostragem, obtém-se amostras de $f(x, y)$ nas direções x e y (domínio) em instantes de tempo discretos, tomadas em um intervalo de tempo uniforme. Este procedimento gera uma matriz de $N \times M$ amostras, digitalizando o plano onde a imagem está inserida (GONZALEZ; WOODS, 2008).

Após a amostragem, faz-se necessário ainda digitalizar a amplitude $z = f(x, y)$ (contradomínio). Para tanto, a amostragem é seguida do uma quantização do valor de f(x,y) em L níveis inteiros de intensidade. Em resumo, como explicam Gonzalez e Woods (2008), a digitalização dos valores de coordenada é chamada de amostragem, enquanto a digitalização dos valores de amplitude é chamada de quantização.

A Figura 13 ilustra a ideia geral dos processos de amostragem e quantização. Uma imagem contínua f, a ser convertida para o formato digital, é apresentada em (a). A função unidimensional apresentada em (b) representa os valores de intensidade da imagem contínua ao longo do segmento de reta AB de (a). Para realizar a amostragem dessa função, amostras igualmente espaçadas são obtidas ao longo da linha AB, indicadas por pequenas linhas verticais na parte inferior da figura em (c). As amostras são representadas por pequenos quadrados brancos sobre a função contínua. Estas localizações discretas descrevem a função de amostragem do eixo horizontal x de f. Para obter uma imagem digital, os valores de intensidade também devem ser convertidos em quantidades discretas. O lado direito da figura em (c) apresenta uma escala de intensidade dividida em oito intervalos discretos, que variam do preto ao branco. Os valores contínuos de intensidade de cada amostra devem ser quantizados atribuindo aos mesmos o mais valor discreto mais próximo dentre os oito. As amostras digitais resultantes dos processos de amostragem e quantização são exibidas em (d). Repetindo este procedimento em outras linhas, tal como no segmento

AB, produz-se uma imagem digital bidimensional. Estas linhas, por sua vez, descreverão a função de amostragem do eixo vertical y de f. Os segmentos de amostragem devem, portanto, ser espaçados de forma uniforme, com o mesmo espaço utilizado na amostragem do eixo x – faz pouco sentido tentar atingir uma densidade de amostragem em uma direção que exceda os limites de amostragem da outra direção.

Figura 13 – Amostragem e quantização de uma imagem contínua



(a) Imagem digital (b) Intensidade ao longo de AB

(c) Amostragem (d) Quantização

Fonte: Adaptada de (GONZALEZ; WOODS, 2008)

3.4 PROCESSAMENTO DIGITAL DE IMAGENS

Segundo Solomon e Breckon (2010), processamento digital de imagens pode ser entendido como o uso de computadores digitais para processar, por meio de algoritmos, imagens digitais. Gonzalez e Woods (2008) apontam duas áreas principais para aplicação de métodos de processamento digital de imagens: melhora das informações visu-

ais para a interpretação humana e processamento de dados de imagens para armazenamento, transmissão e representação, considerando a percepção automática por máquinas.

Esta seção apresenta fundamentos e técnicas de processamento de imagens utilizadas pelo sistema de visão computacional do robô, capaz de reconhecer jogos de Sudoku a partir de imagens digitais e resolvê-los.

3.4.1 Relacionamentos Básicos Entre Pixels

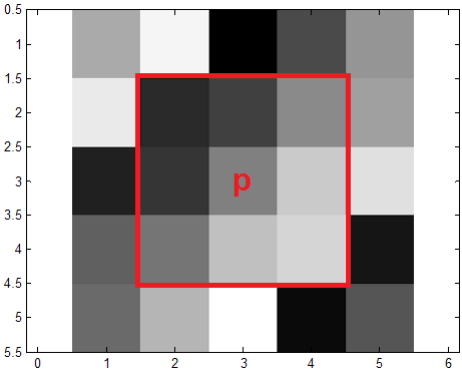
Esta seção apresenta relações entre pixels em uma imagem digital que são de fundamental importância para a compreensão dos demais conceitos apresentados no decorrer deste capítulo.

3.4.1.1 Vizinhança

A vizinhança $N(x, y)$ de um pixel p localizado nas coordenadas (x, y), também chamada de *janela*, é definida como o conjunto de pixels que cercam p, incluindo o próprio p. O formato e o tamanho da vizinhança são definidos de acordo com a aplicação.

Como indicam Gonzalez, Woods e Eddins (2003), o formato mais utilizado é o de janelas quadradas. Neste formato, a vizinhança $N(x, y)$ do pixel p em uma imagem digital f é definida como um quadrado de dimensões ímpares $W \times W = (2M-1) \times (2M-1)$, para $M > 0$ inteiro, centrado em p. A Figura 14 apresenta uma vizinhança na forma de janelas quadradas, com $W = 3$, ao redor de um pixel p em (x, y).

Figura 14 – Vizinhanças em janelas quadradas



Fonte: Adaptada de Gonzalez, Woods e Eddins (2003)

O conceito de vizinhança é utilizado na definição de diversas operações de processamento, tais como filtros, detecção de bordas e operações morfológicas. Outros formatos de vizinhança podem ser utilizados, mas este formato é o mais comum por possuir implementação computacional mais simples.

3.4.2 Operações Espaciais e em Frequência

As operações espaciais são operações realizadas sobre o domínio espacial da imagem. A expressão domínio espacial se refere ao próprio plano imagem, e os métodos de processamento de imagens nesta categoria se baseiam na manipulação direta de pixels em uma imagem (GONZALEZ; WOODS, 2008).

Os processos no domínio espacial podem ser expressos de forma geral pela Equação (3.4), onde $f(x, y)$ é a imagem de entrada, $g(x, y)$ a imagem de saída e T é um operador em f definido em uma vizinhança do pixel em (x, y).

$$g(x, y) = T[f(x, y)] \tag{3.4}$$

Além das operações espaciais, operações no *domínio da frequência* também são amplamente utilizadas no campo de processamento de imagens, como apontam Solomon e Breckon (2010). Nesta classe de operações, a imagem é transformada do domínio do tempo para o

da frequência por meio da transformada de Fourier. Após, operações são realizadas sobre a imagem transformada e, finalmente, realiza-se a transformada inversa para o domínio do espaço, de forma que a imagem possa novamente ser exibida e compreendida por seres humanos. Algumas operações são mais facilmente realizadas no domínio da frequência do que no domínio espacial, motivo pelo qual a transformação é utilizada.

Como todas as operações executadas pelo sistema de visão são realizadas no domínio espacial, este trabalho não irá discorrer sobre operações em frequência. Gonzalez e Woods (2008) pode ser consultado para maiores detalhes sobre o assunto.

3.4.3 Filtragem

Seja $N(x, y)$ uma vizinhança centrada em um pixel arbitrário p de coordenadas (x, y) em uma imagem f. O processamento por vizinhança gera um pixel correspondente nas mesmas coordenadas (x, y) em uma imagem de saída g, de forma que o valor deste pixel é determinado por uma operação específica envolvendo os pixels da imagem de entrada em $N(x, y)$. Este tipo de operação pode ser descrito de forma geral pela Equação (3.5), onde s é a intensidade do pixel no centro de N na imagem processada por T. A este processo, é dado o nome de *filtragem espacial*.

$$s = T(f, N) \tag{3.5}$$

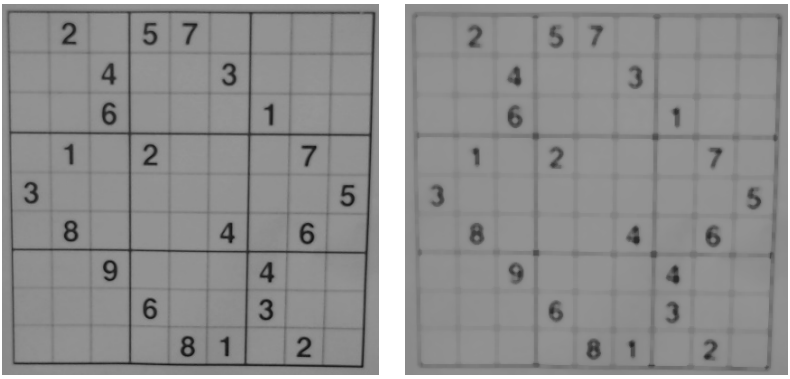
Um exemplo de operação por vizinhança amplamente empregado em processamentos de imagens é o cálculo do valor médio dos pixels em uma vizinhança quadrada de tamanho $M \times M$ centrada em $p = (x, y)$. Esta operação é, de fato, um filtro passa-baixas com janela de tamanho $M \times M$ (SOLOMON; BRECKON, 2010). Esta operação é descrita pela Equação (3.6), onde l e c são as coordenadas de linha e coluna dos pixels da vizinhança $N(x, y)$.

$$m(x, y) = \frac{1}{M \times M} \sum_{(l,c) \in N(x,y)} f(l, c) \tag{3.6}$$

A Figura 15 ilustra o resultado da aplicação do filtro descrito pela Equação (3.6). Uma imagem f em escala de cinza de 8 bits de um jogo de Sudoku é apresentada em *(a)*. O resultado g da aplicação do filtro m sobre esta imagem, utilizando uma vizinhança quadrada

com $M = 7$, é mostrado em *(b)*. A imagem g é criada variando-se as coordenadas (x, y), de forma que o centro da vizinhança se mova de pixel a pixel na imagem f repetindo a operação por vizinhança em cada nova posição.

Figura 15 – Exemplo da operação de filtro de média local



(a) Imagem original (b) Imagem filtrada

Fonte: Elaborada pelo autor

Como é possível observar, o resultado desta operação é um borramento na imagem original. Este filtro é normalmente utilizado para eliminar detalhes e destacar as regiões maiores de uma imagem.

3.4.4 Operações Morfológicas

Segundo Shih (2009), operações morfológicas são operações matemáticas derivadas da teoria de conjuntos que são executadas sobre agrupamentos de pixels, ressaltando aspectos específicos de formas presentes na imagem. As operações fundamentais da morfologia são a *erosão* e a *dilatação*, a partir das quais é possível realizar todas as outras operações.

A aplicação do operador de *erosão* em uma imagem binária encolhe as regiões de foreground, reforçando o background que contorna pixels de foreground. O resultado prático deste operador é a redução do tamanho dos objetos, sem que eles desapareçam. A *dilatação*, por outro lado, tem o efeito contrário a erosão. Quando aplicada a uma imagem binária, esta operação resulta no aumento de tamanho das regiões de

foreground. Desta forma, a dilatação torna os objetos mais largos. A Figura 16 apresenta o resultado destas operações sobre a imagem de um caractere numérico.

Figura 16 – Operações morfológicas de erosão e dilatação

(a) Imagem original (b) Resultado da erosão (c) Resultado da dilatação

Fonte: Elaborada pelo autor

3.4.5 Segmentação

Conforme Sonka, Hlavac e Boyle (2014), segmentação de imagens é o processo de particionar uma imagem nas regiões ou objetos que a compõem. O objetivo da segmentação é identificar objetos ou formas de interesse na imagem, modificando a sua representação de forma conveniente para análise ou operações posteriores.

Para Gonzalez e Woods (2008) a segmentação é uma das mais complexas tarefas no processamento de imagens. Uma segmentação imprecisa pode ocasionar falhas em processos posteriores de análise, acarretando no fracasso do processamento como um todo. Desta forma, todo sistema de processamento de imagens deve maximizar a probabilidade de se obter uma segmentação precisa, com nível de detalhamento suficiente para solucionar o problema ao qual é destinado a resolver.

Esta seção apresenta as técnicas de segmentação utilizadas no sistema de reconhecimento de jogos de Sudoku, apresentado ao final do capítulo, tais como *thresholding* e detecção de linhas.

3.4.5.1 Thresholding

O *thresholding* é o método mais simples de segmentação de imagens, segundo Erthal (2008). Em muitas aplicações de processamento de imagens, o nível de cinza pertencentes a um objeto (*foreground*) são significativamente diferentes do nível de cinza dos pixels de fundo (*background*). O thresholding se torna, então, uma técnica efetiva para separar objetos do fundo.

A Figura 17 apresenta a foto de uma região de um sudoku em escala de cinza de 8 bits. Para que um sistema de processamento de imagens possa reconhecer os dígitos presente nesta imagem, uma das primeiras etapas é separar a grade e os números (*foreground*), objetos de interesse para análise, do fundo da imagem (*background*).

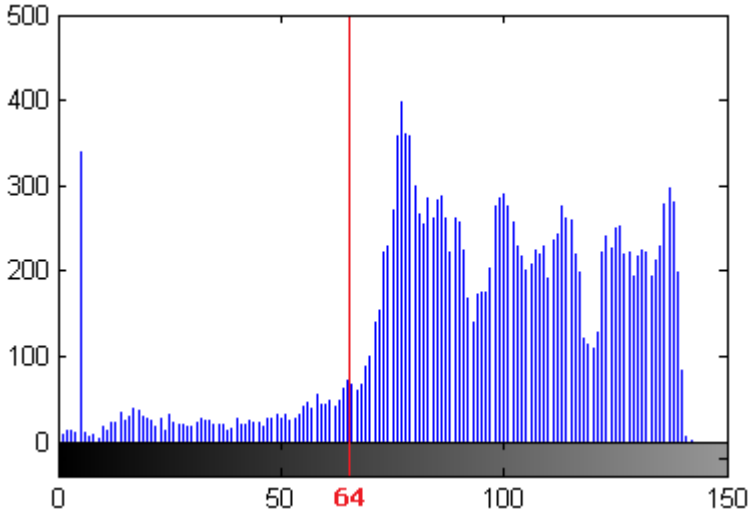
Figura 17 – Região de um sudoku em escala de cinza

Fonte: Elaborada pelo autor

A Figura 18 apresenta o histograma de intensidade desta imagem, $f(x, y)$, composta por objetos escuros sobre um fundo claro de tal forma que os pixels dos objetos e do fundo tenham valores de intensidade agrupados em dois grupos dominantes. Uma maneira de extrair os objetos do fundo é selecionar um limiar de intensidade K que separe estes grupos. Desta forma, qualquer pixel nas coordenadas (x, y) na imagem tal que $f(x, y) < K$ é chamado de ponto de objeto; caso contrário, o ponto é chamado ponto de fundo[1] (ERTHAL, 2008).

[1] Os pixels com intensidade menor que K são considerados de objeto porque os objetos pretos são os de interesse. Se estivéssemos interessados em objetos brancos, seriam os pixels maiores que K

Figura 18 – Histograma de intensidade da região do sudoku



Fonte: Elaborada pelo autor

A imagem segmentada $g(x, y)$ pode então ser definida por meio da Equação (3.7), onde a intensidade '1' representa o foreground e '0' o background. Embora esta convenção seja seguida neste trabalho, quaisquer dois valores que diferenciem background e foreground podem ser utilizados.

$$g(x,y) = \begin{cases} 1 & \text{se } f(x,y) < K. \\ 0 & \text{se } f(x,y) \geq K. \end{cases} \tag{3.7}$$

É importante ressaltar que os grupos de fundo e de objeto devem ser selecionados conforme o problema a ser resolvido. Se o grupo de objetos for claro e o fundo escuro – oposto da situação apresentada –, a relação será oposta, e o grupo de foreground será aquele onde os valores de intensidade são maiores que a constante K. A Figura 19 apresenta o resultado da operação de *thresholding* sobre a imagem da Figura 17, tomando $K = 64$. A saída da operação de thresholding é uma imagem binária na qual um estado representa os objetos de *foreground*, que interessam para a aplicação, e o outro estado corresponde ao *background*.

Figura 19 – Região de um sudoku após thresholding

Fonte: Elaborada pelo autor

Sistemas de análise de documentos por imagem incluem diversas tarefas de processamento de imagem, iniciando pela digitalização do documento até o reconhecimento de caracteres. A binarização da imagem, realizada comumente por uma técnica de thresholding, é normalmente a primeira tarefa após a aquisição na maioria destes sistemas, como destacam Borovikov e Lane (2004). Esta representação é particularmente conveniente, uma vez que a maioria dos documentos possuem apenas uma cor para o texto e uma outra cor diferente para o fundo. Ainda, tal representação diminui a carga computacional e permite a utilização de métodos mais simples para análise posterior, se comparados aos utilizados para imagens coloridas e em tons de cinza.

Kieri (2012) mostra que a eficiência da técnica de thresholding aplicada pode afetar criticamente as etapas posteriores de um sistema de visão, tais como a identificação de objetos e o reconhecimento de caracteres. Segundo Sezgin e Sankur (2004), deformações no formato dos caracteres, como consequência de um thresholding mal sucedido, são as principais razões para o insucesso de técnicas de OCR.

A grande dificuldade no processo de thresholding é estabelecer o limiar K para separar os objetos do fundo. Iluminação, semelhança entre níveis de cinza de objeto e background, contraste inadequado e o ruído são alguns dos fatores que podem tornar as distribuições de pixels de fundo e dos objetos de interesse insuficientemente diferentes, impossibilitando o estabelecimento de um único limiar global aplicável a toda imagem. Além disso, variações destes fatores ocorrem de imagem para imagem caso o ambiente não seja controlado, impedindo que o sistema de processamento estabeleça um limiar fixo igual para todas as imagens de entrada (DAVIES, 2012).

Para tornar o thresholding um processo completamente automá-

tico, é necessário que o sistema de processamento de imagens selecione o limiar K adequado para cada imagem a ser processada. Sezgin e Sankur (2004) conduziram uma pesquisa com 40 métodos diferentes de thresholding, comparando seu desempenho e categorizando-os posteriormente em seis grupos distintos baseados na forma de cálculo do limiar K. Os resultados desta pesquisa apontam que uma destas seis categorias, a dos *métodos locais adaptativos* (*Adaptive Local Thresholding*), possuem o melhor desempenho em aplicações de análise de documentos.

3.4.5.2 Adaptive Local Thresholding

As técnicas de análise de documentos requerem que o conteúdo lógico e semântico seja preservado durante a operação de thresholding. Haja vista que a degradação resultante deste processo é uma das maiores razões para insucessos no processamento (SEZGIN; SANKUR, 2004), é importante utilizar uma técnica de binarização que detecte e filtre possíveis imperfeições para que não provoquem erros em etapas seguintes. Este requisito impede o uso de um threshold global em muitos casos. As principais situações em que um único threshold global não é suficiente são onde existem gradientes de iluminação, baixa qualidade da imagem do documento e complexidade na estrutura do documento (DU et al., 2013).

Enquanto operações convencionais de thresholding utilizam um valor global de limiar para todos os pixels, os métodos adaptativos locais selecionam o valor de limiar dinamicamente, com base na vizinhança de cada pixel (SEZGIN; SANKUR, 2004). Assim, um K diferente é calculado para cada pixel da imagem. Estes métodos são mais sofisticados e capazes de desempenhar um bom papel mesmo com gradientes de iluminação na imagem, o que os torna ideais para aplicações de análise de documentos.

Existem diferentes abordagens para encontrar o limiar local em métodos adaptativos. A maioria examina os valores de intensidade dos pixels da vizinhança quadrada $M \times M$ centrada em cada pixel. A estatística mais apropriada aplicada sobre a vizinhança depende do tipo de imagem de entrada. A mais utilizada, no entanto, é a operação de média de intensidade dos pixels (SEZGIN; SANKUR, 2004). Além da estatística, uma tolerância C é normalmente considerada no cálculo do limiar, com o objetivo de filtrar uma parte do ruído. A Equação (3.8) apresenta a operação descrita, onde $m(x, y)$ representa a média de intensidade da vizinhança quadrada com centro em (x, y), M é o

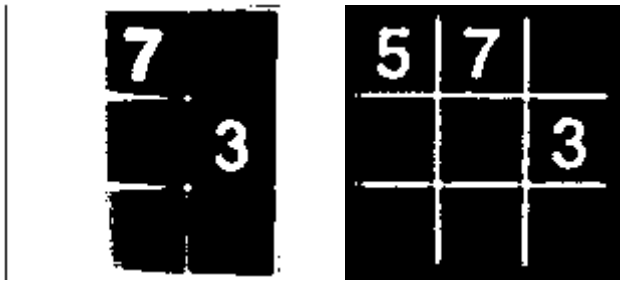
tamanho do lado da janela da vizinhança e C é a tolerância. Este método é conhecido como *método de Sauvola* (SAUVOLA; PIETIKäINEN, 2000).

$$T_M(x, y) = m(x, y) - C \tag{3.8}$$

O valor de tolerância C pode ser determinado de diversas formas. Sauvola e Pietikäinen (2000) sugere a utilização do desvio padrão relativo da vizinhança, multiplicado por um valor de *bias*. Por simplicidade, no entanto, a tolerância C é normalmente definida como um valor constante obtido empiricamente.

A Figura 20 ilustra a diferença entre os thresholdings global e adaptativo. O resultado da limiarização global ótima sobre a imagem da Figura 17 é apresentada em (a). O *método de Otsu*, um dos mais eficientes métodos para determinação do limiar global (SEZGIN; SANKUR, 2004), foi utilizado. O resultado da aplicação do método de binarização adaptativa local de Sauvola é apresentado em (b) utilizando vizinhanças quadradas de tamanho 11×11 e $C = 4$. Percebe-se claramente que o método de Sauvola foi melhor que o de Otsu, preservando os objetos de interesse na imagem ao realizar a limiarização. A limiarização pelo método de Otsu, por outro lado, resultou na perda de informações de fundamental importância para a aplicação.

Figura 20 – Thresholding global (Otsu) e local (Sauvola) sobre região do sudoku



(a) Método de Otsu (b) Método de Sauvola

Fonte: Elaborada pelo autor

A grande desvantagem dos métodos de threshold adaptativos locais é sua performance, que é geralmente mais lenta se comparada aos métodos de threshold global (SEZGIN; SANKUR, 2004). Isso se deve ao

fato de que a computação nestes métodos é realizada para a vizinhança local de cada pixel da imagem. O cálculo de média de vizinhanças quadradas de dimensões $M \times M$ para uma imagem de dimensões L de largura e H de altura, por exemplo, necessita de aproximadamente $L \times H \times M^2$ operações de soma e $L \times H$ divisões por M^2. Felizmente, é possível melhorar o desempenho do cálculo da média de vizinhanças utilizando o conceito de *tabelas de soma*, ou *imagens integrais*, como mostram Shafait, Keysers e Breuel (2008). A implementação utilizando o conceito de imagem integral reduz significativamente a quantidade de operações realizadas.

3.4.5.3 Detecção de Linhas

Detecção de linhas é o processo de segmentação empregado para identificar agrupamentos de pixels de foreground que formam linhas retas, ou aproximadamente retas, em uma imagem de interesse (GONZALEZ; WOODS; EDDINS, 2003). Existem diversos métodos de detecção de linhas, que variam quanto a complexidade computacional, eficiência e robustez. Um dos métodos mais utilizados em sistemas de processamento de imagens é a avaliação de máximos locais da *transformada de Hough* (GONZALEZ; WOODS, 2008). Trata-se de um método tolerante a falhas de descrição de fronteiras de objetos e que é pouco afetado por ruídos nas imagens.

A *transformada de Hough* é uma técnica que pode ser utilizada para identificar objetos com formas facilmente parametrizadas numa imagem – linhas, elipses e círculos, por exemplo –, por meio de um procedimento de votação. Este procedimento consiste em variar os parâmetros de um espaço de parâmetros conveniente em busca de objetos candidatos. Os objetos candidatos são pontuados (votados) de forma proporcional a quantidade de vezes que objetos descritos por seus parâmetros são localizados. Os pontos de máximo local na função que descreve o resultado do procedimento de votação identificam, assim, objetos com grande probabilidade de possuir a forma de interesse buscada na imagem (GONZALEZ; WOODS; EDDINS, 2003).

O caso mais simples da transformada de Hough é justamente a detecção de linhas retas, que funciona como segue. A transformada opera sobre uma imagem binária, percorrendo todos os pixels da imagem. Quando um pixel de foreground é encontrado, a transformada desenha as possíveis retas que passam por este pixel, na forma de linhas virtuais. As linhas virtuais são desenhadas variando parâmetros

da equação da reta em passos discretos, com uma resolução suficiente para detectar as formas de interesse. O tamanho dos passos utilizados é determinado de acordo com a aplicação, dependendo do nível de detalhes que a ser obtido.

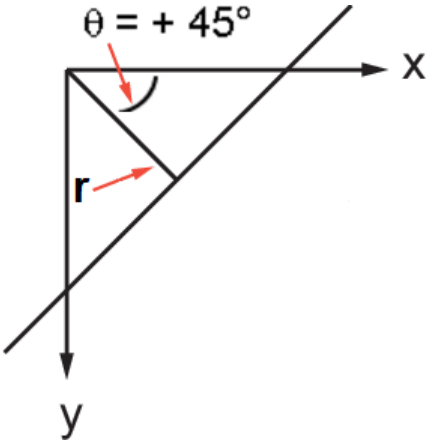
Em geral, a Equação (3.9) é utilizada para representar retas, possuindo como parâmetros o coeficiente angular a e o coeficiente linear b. O problema desta representação é que a se aproxima do infinito ao passo em que uma reta se aproxima da inclinação vertical, dificultando sua representação em um espaço de parâmetros.

$$y = ax + b \tag{3.9}$$

Para contornar este empecilho, Duda e Hart (1972) propuseram utilizar a forma normal (coordenadas polares) para representar as linhas virtuais, descrita pela Equação (3.10), onde r é a distância da origem ao ponto mais próximo da reta e θ é o ângulo entre o eixo x e este ponto. Os parâmetros x e y são as coordenadas do ponto de cada ponto analisado. O espaço de parâmetros passa a ser então o plano $r\theta$, com θ variando de acordo com o tamanho do passo definido. A ideia é apresentada na Figura 21.

$$r = x\cos(\theta) + y\sin(\theta) \tag{3.10}$$

Figura 21 – Transformada de Hough utilizando coordenadas polares



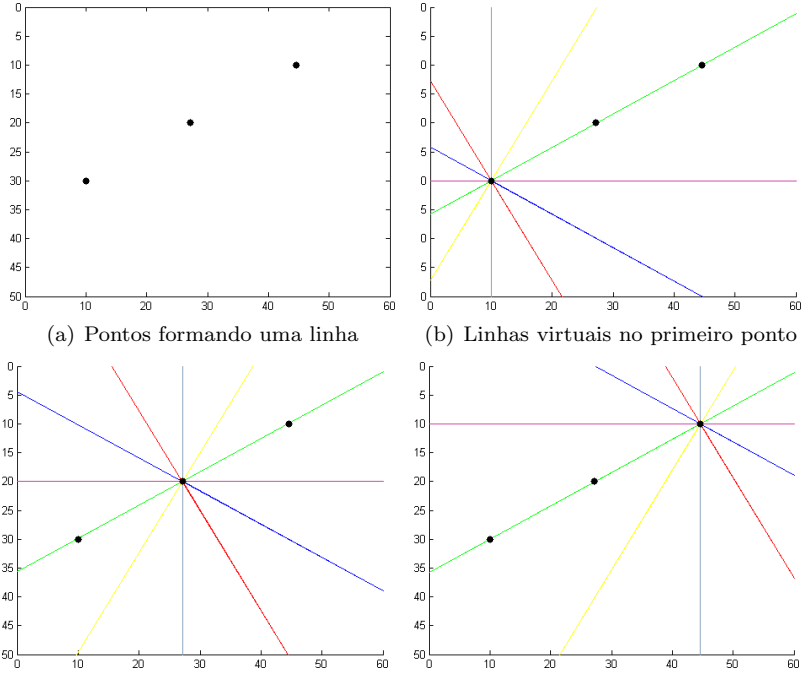
Fonte: Elaborada pelo autor

Para cada combinação de parâmetros que descreve uma reta,

existe um contador chamado de acumulador (GONZALEZ; WOODS; EDDINS, 2003). Os acumuladores são definidos por meio da divisão do espaço de parâmetros $r\theta$ em *células acumuladoras*, onde cada célula representa uma fração deste espaço. Se o ponto no plano $r\theta$ descrito parâmetros da linha virtual candidata incidir na área coberta por uma célula acumuladora, o acumulador desta célula é incrementado. Este incremento, que caracteriza o processo de votação, aumenta a probabilidade da linha virtual ser uma linha real. Assim, linhas reais tendem a possuir a maioria dos votos ao final do procedimento. As células acumuladores são representadas computacionalmente por meio de um vetor de inteiros com $N-$dimensões, onde N é o número de parâmetros do espaço de parâmetros.

Na Figura 22 é possível observar em (a) uma sequência de pontos que apresenta disposição linear. Para identificar a linha sob a qual os pontos estão inscritos, a transformada de Hough traça linhas virtuais em diferentes ângulos θ que passam por cada um destes pontos e distam da origem por um dado r. Estas linhas virtuais são apresentadas em (b), (c) e (d), sendo cada ângulo representado por uma cor diferente. O único caso em que um mesmo conjunto $r\theta$ descreve uma linha por mais de uma vez é aquele em que a linha passa sobre os três pontos – representado pela linha verde nas três imagens. Assim, este conjunto $r\theta$ definiria um máximo local e a linha dos pontos seria identificada com a transformada de Hough.

Figura 22 – Transformada de Hough para linhas retas



(a) Pontos formando uma linha

(b) Linhas virtuais no primeiro ponto

(c) Linhas virtuais no segundo ponto

(d) Linhas virtuais no terceiro ponto

Fonte: Adaptada de Sonka, Hlavac e Boyle (2014)

A transformada de Hough foi concebida para detectar linhas retas. Seu conceito foi estendido posteriormente para detectar também outros objetos de formas regulares, como retângulos, círculos e elipses. Como a detecção de demais formas não se faz necessária para o desenvolvimento da aplicação final, este trabalho se restringirá a apresentação da transformada de Hough apenas para linhas retas.

3.5 RECONHECIMENTO ÓPTICO DE CARACTERES

Segundo Borovikov e Lane (2004), *Reconhecimento Óptico de Caracteres*, amplamente conhecido pela sigla OCR do inglês *Optical Character Recognition*, é o processo de converter caracteres presentes em imagens digitais para o formato de um arquivo de texto editável por computador. A saída deste processo deve ser, idealmente, o texto

presente na imagem. O OCR é um método comum de digitalizar textos impressos para que estes possam ser editados, indexados para pesquisa, armazenados de forma mais compacta, exibidos online ou utilizados como entrada em algum processador de textos, tais como sistemas *text-to-speech* e de mineração de texto.

O OCR é um campo de pesquisa das áreas de reconhecimento de padrões, inteligência artificial e visão computacional, e se apresenta como um problema desafiador que ainda não foi completamente solucionado, apesar de bastante desenvolvido. Mesmo o reconhecimento de caracteres impressos não é uma tarefa simples, haja vista que existem variações de um mesmo caracter provocadas por mudanças de fonte, tamanho ou diferentes tipos de ruído. Técnicas de OCR do atual estado da arte possuem taxas de acerto próximas de 99% em imagens de documentos impressos e de alta qualidade (NIKLAS, 2010). Assumindo uma média de 5 letras por palavra, isso significa que uma em cada 20 palavras não são reconhecidas corretamente, ou seja, pelo menos 5% de todas as palavras processadas possuirão erros. Em textos manuscritos ou degradados, a taxa de erros é ainda maior.

Segundo Borovikov e Lane (2004), o processo de reconhecimento óptico de caracteres divide-se em três etapas: pré-processamento; reconhecimento e; pós-processamento. Diversas técnicas são aplicadas em cada etapa para que os caracteres presentes nas imagens sejam reconhecidos ao fim do processo. As seções seguintes detalham estas etapas e algumas das técnicas mais populares empregadas em sistemas de reconhecimento de caracteres modernos.

3.5.1 Pré-processamento

Antes de realizar a etapa de reconhecimento propriamente dita, é comum que sistemas de reconhecimento de caracteres executem operações de pré-processamento nas imagens de entrada . Estas operações são realizadas com o intuito de reforçar as características dos caracteres analisados, aumentando as chances de sucesso no reconhecimento.

As principais operações utilizadas na etapa de pré-processamento de um sistema moderno de OCR são apresentadas nas subseções seguintes.

3.5.1.1 Binarização

A binarização é o processo de converter uma imagem colorida ou em escala de cinza para sua representação binária. Esta operação é realizada com o intuito de separar o texto dos demais elementos da imagem. Executa-se a binarização em praticamente todos os sistemas de OCR, uma vez que a maioria dos algoritmos de reconhecimento de caracteres trabalham apenas com imagens binárias (SEZGIN; SANKUR, 2004).

Kieri (2012) mostra que a eficiência da operação de binarização influencia significativamente o desempenho da etapa de reconhecimento. Por este motivo, a segmentação do texto é uma operação crítica no reconhecimento de caracteres. Normalmente, os analistas selecionam o método de binarização utilizado no sistema de acordo com a aplicação, visto que a qualidade da binarização depende do tipo de imagem a ser processada. Um documento histórico manuscrito degradado, por exemplo, deve ser tratado de forma diferente de um documento impresso em boas condições.

3.5.1.2 Ajuste de Inclinação

É comum que os documentos não fiquem alinhados perfeitamente com o hardware de aquisição quando são capturados por câmeras, apresentando uma rotação no sentido horário ou anti-horário. Esta rotação pode dificultar, ou até mesmo impossibilitar, o processo de reconhecimento dos caracteres. Para solucionar tal problema, os sistemas de reconhecimento de caracteres detectam o ângulo de rotação do documento na etapa de pré-processamento e o corrigem, de forma que o texto fique perfeitamente alinhado horizontal e verticalmente e possa ser então reconhecido.

A operação de detectar e corrigir o ângulo de rotação do texto na imagem é chamada de *ajuste de inclinação* (NAZ et al., 2013). Existem diversas técnicas diferentes para realizar esta operação. Uma das mais comuns é detectar linhas de referência no texto, calcular o ângulo das mesmas e rotacionar toda a imagem com este ângulo, porém no sentido contrário. As linhas de referência podem ser detectadas por meio da transformada de Hough. Naz et al. (2013) mostra que mesmo em textos sem pauta, as letras tendem a ficar alinhadas na parte inferior e superior, criando um padrão linear que é detectado pela transformada.

O ajuste de inclinação em jogos de Sudoku também pode ser

realizado utilizando a transformada de Hough. Neste caso, as linhas da grade são obtidas como resultado da transformada e servem como referência para calcular o ângulo de rotação do jogo na imagem.

3.5.1.3 Filtragem

Outra técnica amplamente empregada no pré-processamento é a aplicação de filtros, utilizados para acentuar as características dos objetos de interesse na imagem (GONZALEZ; WOODS, 2008). Filtros são muito úteis na redução de ruídos, que são um dos maiores obstáculos para os algoritmos de reconhecimento de caracteres (DU et al., 2013).

Os filtros utilizados variam de acordo com a aplicação. Filtros passa-baixas são utilizados com frequência, visto que reduzem consideravelmente os ruídos de alta frequência ao passo que mantém as características gerais dos caracteres (GONZALEZ; WOODS; EDDINS, 2003).

3.5.1.4 Segmentação de Caracteres

Os algoritmos de reconhecimento de caracteres, como o nome sugere, possuem como entrada imagens de caracteres individuais[2]. Desta forma, é necessário que o texto sob análise seja separado nos caracteres que o constituem durante a etapa de pré-processamento. Esta tarefa é realizada pela operação de *segmentação de caracteres*.

O processo de segmentação de caracteres é responsável por separar caracteres que estão conectados no texto. Ainda, esta operação deve unir caracteres únicos que estão separados em múltiplas partes devido a falhas no texto impresso ou na imagem obtida, como aponta Shih (2009).

No reconhecimento dos números presentes em jogos de Sudoku, a separação de caracteres é facilitada pela divisão da grade em células bem definidas e separadas.

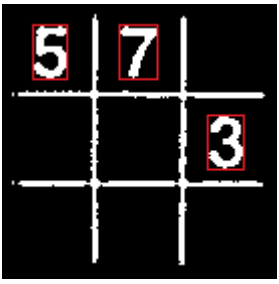
[2]Existem também métodos que não segmentam os caracteres para reconhecimento, mas eles são conhecidos como *Reconhecimento Inteligente de Palavras*, uma classe sucessora dos métodos de OCR. Ravani, Nooralishahi e Amani (2011) apresenta detalhes.

3.5.1.5 Enquadramento

O sucesso de algoritmos de reconhecimento baseados na segmentação de caracteres depende diretamente da eficácia desta separação (BOROVIKOV; LANE, 2004). Para que a separação dos caracteres seja realizada com sucesso em imagens complexas, localizar previamente as regiões candidatas a conterem caracteres é de grande utilidade. Este processo, chamado de *enquadramento*, divide a imagem em pequenas regiões para análise individual, reduzindo os problemas relacionados à grande variedade de textura, objetos e cores presentes na imagem como um todo (BOROVIKOV; LANE, 2004).

O enquadramento localiza as regiões candidatas a conter caracteres na imagem e as delimita por meio uma caixa retangular, conhecida na literatura como *bounding box*. Ao mesmo tempo em que uma *bounding box* deve enquadrar todo o caractere, garantindo que nenhuma informação seja perdida, ela deve também possuir o menor tamanho possível. Esta condição, aliada a técnica de normalização, permite que caracteres de diferentes tamanhos sejam reconhecidos. A Figura 23 apresenta números em células de um jogo de Sudoku enquadrados.

Figura 23 – Enquadramento de caracteres



Fonte: Elaborada pelo autor

Borovikov e Lane (2004) ressalta que além de localizar regiões candidatas, o enquadramento também deve verificar quais destas regiões são, de fato, regiões textuais. Este procedimento é realizado por meio da extração de atributos de cada *bounding box* obtida na etapa de localização e, mediante a avaliação desses atributos, elas são classificadas como textuais ou não-textuais. Apenas as regiões textuais são posteriormente enviadas para a classificação.

3.5.1.6 Normalização

A maioria dos algoritmos utilizados para reconhecer caracteres em imagens digitais compara as entradas fornecidas com templates previamente armazenados dos possíveis caracteres, como aponta o levantamento realizado por Du et al. (2013). Os templates são amostras de imagens digitais destes caracteres, obtidas com antecedência e armazenadas no sistema de reconhecimento. A classificação é realizada comparando a imagem de entrada com estes templates: a imagem de entrada é classificada como o caractere que possuir o template mais semelhante a ela.

Uma vez que o tamanho dos caracteres muda de texto para texto – e em alguns casos até dentro de um mesmo texto –, as regiões obtidas durante o enquadramento precisam ser redimensionadas antes de ser processadas pelo classificador, de modo que possuam o mesmo tamanho dos templates de caracteres. A normalização consiste em redimensionar as *bounding boxes* de forma que todas apresentem o mesmo tamanho dos templates de referência utilizados para comparação pelo classificador. Esta técnica permite que caracteres de diferentes tamanhos sejam reconhecidos pelo algoritmo de OCR (BOROVIKOV; LANE, 2004).

A proporção P em que cada região deve ser redimensionada é dada pela razão entre as dimensões (altura h e largura L) do template T e da *bounding box* BB, conforme Equação (3.11).

$$P_h = \frac{h_T}{h_{BB}}, \qquad P_L = \frac{L_T}{L_{BB}} \tag{3.11}$$

3.5.2 Reconhecimento

É na etapa de *reconhecimento* que os caracteres presentes nas imagens são de fato reconhecidos. Como a classificação de caracteres é um dos campos de teste favoritos para a aplicação de novas ideias de reconhecimento de padrões (BOROVIKOV; LANE, 2004), diversas técnicas diferentes foram desenvolvidas ao longo dos anos para solucionar este problema. Apesar da variedade de abordagens existentes, os reconhecedores de uma forma geral se dividem em dois estágios: *extração de características* e *classificação*.

Na *extração de características*, a imagem resultante das operações de pré-processamento é recebida e são obtidas informações relevantes (descritores) sobre objetos presentes na mesma. Estas informações são

então utilizadas como entrada para o estágio de *classificação*, que determina a partir destes descritores qual é o caractere na imagem. Além de classificar a entrada em um caractere conhecido, os classificadores informam também um nível de confiança, ou seja, um indicador do quão certo estão sobre a classificação realizada.

Du et al. (2013) e Borovikov e Lane (2004) realizaram levantamentos sobre métodos de reconhecimento utilizados em sistemas modernos de OCR. Os principais métodos elencados por ambos são apresentados nas próximas seções.

3.5.2.1 Template Matching

O *Template Matching*, também conhecido como *Matrix Matching*, é um dos métodos mais populares de classificação de caracteres e foi um dos primeiros a ser desenvolvido para realizar esta tarefa (DU et al., 2013). Este método compara a imagem de entrada com um conjunto de *templates* (imagens) de todos os caracteres possíveis, previamente amostrados e armazenados no sistema de reconhecimento.

A classificação é realizada comparando a intensidade de cada pixel da imagem de entrada $I(x, y)$ com a intensidade do pixel equivalente de todos os templates de caracteres $T_C(x, y)$ conhecidos pelo sistema. Cada comparação resulta em uma medida de diferença s entre o caractere de entrada e o template. A diferença é menor quando o pixel da imagem de entrada é semelhante ao pixel que está na mesma localização na imagem de template, ou seja, quando ocorre um *match*. Quando os pixels correspondentes são diferentes, ou seja, ocorre um *mismatch*, a medida de diferença aumenta. O caractere analisado é classificado como aquele em que o template possui menor diferença com a imagem de entrada.

A função $s(I, T_C)$ utilizada para realizar a comparação e obter a diferença entre imagem e templates varia de sistema para sistema. Muda et al. (2007) aponta que as mais utilizadas, no entanto, são as funções de *distância de Manhattan*, *distância euclideana*, *correlação cruzada* e *correlação normalizada*. A distância de Manhattan entre uma imagem $I(x, y)$ e o template de um caractere $T_C(x, y)$, de largura l e altura h, é dada pela Equação (3.12).

$$s(I, T_C) = \sum_{i=0}^{l-1} \sum_{j=0}^{h-1} \mid I(i, j) - T_C(i, j) \mid \tag{3.12}$$

As operações de enquadramento e normalização desempenham um papel fundamental para que os algoritmos de reconhecimento baseados em *Template Matching* tenham sucesso (BOROVIKOV; LANE, 2004). Caso estas operações não sejam bem sucedidas, os caracteres ficarão com tamanho e posição diferentes a cada amostra obtida, inviabilizando uma comparação razoável com os templates de referência utilizados pelo classificador.

Os templates utilizados pelo sistema de classificação são obtidos por meio de amostras de caracteres pertencentes a um conjunto de treinamento. O desenvolvedor do sistema de reconhecimento toma amostras de cada caractere que possa estar presente nos documentos reconhecidos pelo classificador e cria uma base de comparação com estas amostras. Para que o classificador seja mais preciso, as amostras dos caracteres de treinamento devem ser tão similares quanto possível às imagens de caracteres que serão fornecidas como entrada para o classificador posteriormente.

Ainda, para que a classificação obtenha melhores resultados, os templates não devem ser formados por uma única amostra de treinamento: diversas amostras devem compor o template de referência (SONKA; HLAVAC; BOYLE, 2014). O template é normalmente formado tomando o valor médio de intensidade para cada pixel das imagens que compõem o conjunto de treinamento. Classificadores treinados com um conjunto de treinamento pequeno podem reconhecer com grande confiança alguns caracteres e, ao mesmo tempo, falhar em reconhecer estes mesmos caracteres caso eles possuam uma pequena diferença. Para tornar os classificadores mais robustos, o conjunto de treinamento pode possuir o mesmo caractere escrito em diferentes fontes e com pequenas rotações.

Apesar de sua simplicidade, este método é muito eficaz quando aplicado a textos que utilizam uma mesma fonte, como mostra Du et al. (2013).

3.5.2.2 Análise Topológica

A *Análise Topológica* classifica os caracteres de acordo com as suas características estruturais. Estas características podem ser definidas em termos de linhas, formas geométricas, buracos, ou outros atributos, como cantos e concavidades (SONKA; HLAVAC; BOYLE, 2014). O número '8', por exemplo, pode ser descrito como "dois círculos tangentes entre sí, alinhados verticalmente".

As características extraídas em métodos baseados na análise topológica podem variar. As mais comuns são altura e largura do caractere, linhas, formas fechadas, cantos, concavidades, interseções de linhas e outros traços que possam ser relevantes na classificação do caractere (DU et al., 2013).

A classificação é realizada verificando a presença ou a ausência de traços característicos dos caracteres, por meio de um classificador que compara informações extraídas da estrutura da imagem de entrada com as descrições de cada caractere conhecido pelo sistema, armazenadas em uma base de conhecimento. A comparação pode ser realizada baseando-se em regras abstratas ou por meio de um algoritmo de classificação em reconhecimento de padrões (IMPEDOVO; OTTAVIANO; OCCHINEGRO, 1991). Os algoritmos mais utilizados são os classificadores de vizinhos mais próximos, como o *k-nearest neighbors* (*k-NN*) (DU et al., 2013).

Além de ser muito eficiente quando aplicado a imagens de alta qualidade, este método também possui bom desempenho em imagens de baixa qualidade.

3.5.2.3 Métodos Híbridos

Outra técnica muito utilizada em sistemas modernos de reconhecimento de caracteres, segundo Du et al. (2013), é combinar dois ou mais métodos distintos para realizar o reconhecimento. Naturalmente, esta técnica exige maior tempo de processamento, porém a combinação tende a possuir uma taxa de acertos maior que a dos classificadores utilizados individualmente.

3.5.2.4 Outros métodos

Diversos outros métodos também podem ser empregados para realizar o reconhecimento de caracteres. O sucesso do método utilizado depende diretamente do tipo de aplicação: cada um pode apresentar resultados melhores ou piores dependendo da situação.

Du et al. (2013) apresenta um levantamento detalhado de técnicas de OCR utilizadas atualmente. Dentre alguns dos métodos levantados, destacam-se um baseado em Modelos Ocultos de Markov; outro que utiliza Máquinas de Vetores de Suporte na configuração multiclasse e; um classificador desenvolvido com regras de lógica fuzzy.

3.5.3 Pós-processamento

A eficiência do reconhecimento de caracteres pode ser aumentada se a saída da etapa anterior for processada por um analisador léxico, que verifique se as palavras reconhecidas na etapa anterior são permitidas no contexto onde foram encontradas (BOROVIKOV; LANE, 2004). Um analisador pode, por exemplo, consultar um dicionário da língua em que o texto foi escrito, verificando se cada palavra reconhecida consta no mesmo e efetuando as correções necessárias nos casos em que uma palavra for desconhecida, substituindo-a pela palavra com grafia mais próxima encontrada neste dicionário.

Sistemas de reconhecimento de caracteres modernos podem possuir uma etapa de pós-processamento, que é normalmente adaptada para o tipo específico de entrada ao qual o sistema se destina a processar. Esta estratégia é chamada de *OCR Orientado a Aplicação* ou *OCR Customizado* e tem sido utilizada em aplicações como o reconhecimento de placas veiculares, análise de cartões de visita e digitalização de documentos históricos .

No reconhecimento de jogos de Sudoku, o pós-processamento pode ser útil para aumentar a taxa de acertos dos números identificados na etapa anterior baseando-se nas regras do jogo, verificando se nenhuma regra foi violada na grade montada durante o reconhecimento. É comum, por exemplo, que o número 9 seja reconhecido incorretamente como o número 8, fato que acontece pela semelhança dos números em questão. A existência de dois números iguais nas células de mesma linha, coluna ou região, no entanto, evidencia que o reconhecimento de pelo menos um destes caracteres falhou. Se isto acontecer, o sistema pode proceder com uma correção, substituindo o valor da célula que teve a menor probabilidade de acerto entre as duas pelo segundo número mais provável para esta célula e que não viole as regras do jogo.

4 FUNDAMENTOS DE ROBÓTICA

O *Instituto Americano de Robótica* define robô como um *"manipulador reprogramável e multi-funcional projetado para mover materiais, partes, ferramentas ou dispositivos especializados através de movimentos variáveis programados para desempenhar uma variedade de tarefas"* (CHIAVENATO, 1983). Tipicamente, robôs são compostos por uma estrutura mecânica; um conjunto de sensores; acionadores ou atuadores; uma fonte de energia elétrica e; um computador que controla e coordena todo o sistema. Desta forma, os projetos mecânico e do sistema de controle por computador caracterizam-se como etapas essenciais para a construção de um robô.

Este capítulo apresenta os fundamentos de robótica, incluindo aspectos mecânicos e de controle, necessários para a construção do robô solucionador de Sudoku, descrito em detalhes no capítulo seguinte.

4.1 O QUE É UM ROBÔ

Segundo Merriam-Webster Online (2009), robô é *"um dispositivo automatizado capaz de manipular objetos ou de executar operações de acordo com um programa fixo, modificável ou adaptável"*. Existem, no entanto, vários outros conceitos e definições para a palavra robô, que decorrem em razão da diversidade de configurações, funções e aplicações desses dispositivos automatizados. Devido a esta diversidade, robôs são comumente classificados de acordo com suas aplicações e formas de trabalhar (ROSARIO, 2010). Uma das principais classes de robôs são os *robôs industriais*, que se refere aos robôs utilizados na indústria.

Atualmente, a maior aplicação de robôs é na área industrial, principalmente na produção de bens de consumo (ROSARIO, 2010). A norma *ISO 8373* define um *robô industrial* como um *"manipulador multipropósito controlado automaticamente, reprogramável, programável em três ou mais eixos"* (ISO, 2012). Os robôs industriais são aqueles desenvolvidos tipicamente para uso nos processos de manufatura. Estes processos incluem, mas não se limitam, a realização de cortes, furos, soldagens e montagens de produtos finais.

Existem diversos outros tipos de robôs além dos industriais. Como exemplo, pode-se citar os *rovers* e os *androides*. Este trabalho, no entanto, se restringirá a falar apenas sobre robôs industriais e, desta forma, a palavra robô será utilizada a partir deste ponto para se

referir exclusivamente aos robôs desta classe.

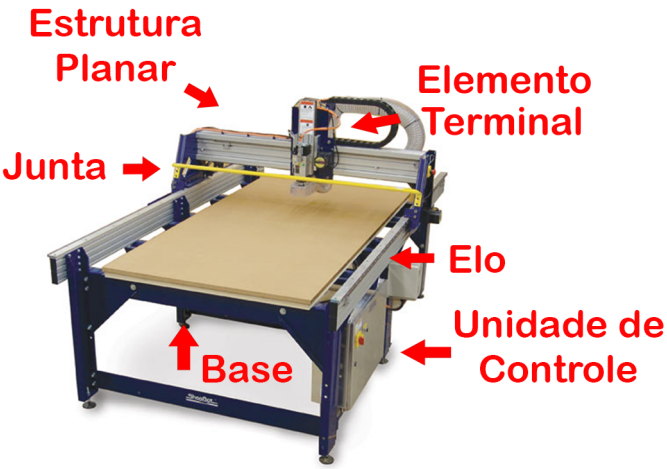
4.2 CARACTERÍSTICAS GERAIS

Os robôs cartesianos são os mais comumente empregados em máquinas industriais, tais como tornos e centros de usinagem (CAMPION; WANG; HAYWARD, 2005). Estas máquinas, normalmente programáveis por meio comando numérico computadorizado (CNC), possuem alta precisão e são utilizadas para realizar operações de corte, fresagem, maquinação e afins.

Robôs cartesianos possuem quatro componentes fundamentais: uma base, a qual pode girar ou permanecer fixa; uma estrutura planar cartesiana, utilizada para mover o elemento terminal do robô; uma unidade de controle, ou seja, o controlador do robô e; um dispositivo de programação (CORKE, 2011). O nome *cartesiano* é dado em razão da estrutura planar (ou cúbica) para movimentação do elemento terminal nesta configuração mecânica.

De uma forma geral, a estrutura mecânica de um robô é construída como uma junção de elos, arranjados para mover um elemento terminal (CORKE, 2011). As articulações, que conectam os elos, permitem que estes se movimentem.

Figura 24 – Exemplo de robô cartesiano



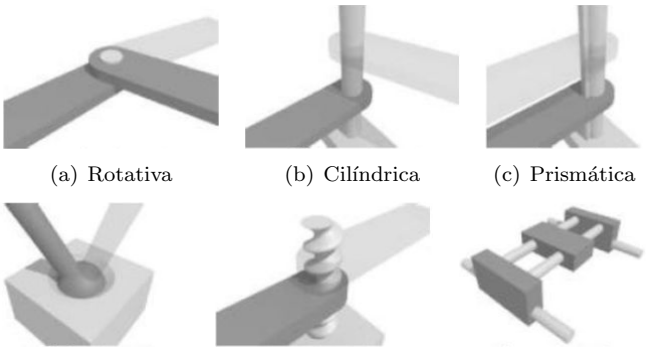
Fonte: Adaptado de PRSalpha (2015)

Os movimentos do robô são realizados por motores, que controlam a posição das juntas apoiados por dados provenientes de sensores. O elemento terminal é o responsável pela ação final da máquina, constituindo-se como a ligação entre o robô e o meio em que ela atua. A Figura 24 ilustra um exemplo de robô cartesiano, destacando suas partes fundamentais.

4.2.1 Articulações

As *articulações*, comumente chamadas de *juntas*, conectam dois elos (ou mais) e são responsáveis pelo movimento entre eles (ROSARIO, 2010). Existem diversos tipos de juntas, utilizadas para realizar diferentes tipos de movimentos. As juntas podem ser dos tipos *rotativa*, *prismática*, *cilíndrica*, *esférica*, *parafuso* e *planar*. As juntas rotativas e prismáticas, no entanto, são as mais utilizadas em robôs, visto que o movimento de todas as outras juntas pode ser derivado a partir delas (ROSARIO, 2010). A Figura 25 apresenta os tipos de juntas utilizados em robôs.

Figura 25 – Tipos de juntas utilizadas em robôs



(a) Rotativa (b) Cilíndrica (c) Prismática

(d) Esférica (e) Parafuso (f) Planar

Fonte: Adaptada de Marcato (2012)

As *juntas rotativas* realizam um movimento rotacional em torno de um eixo imaginário estacionário – o eixo de rotação. Seu funcionamento pode ser comparado ao de uma dobradiça. O movimento realizado por uma junta deste tipo é função do ângulo de abertura da mesma.

As *juntas prismáticas*, por sua vez, realizam movimento linear sobre um eixo estacionário, sem girar. São compostas de duas hastes que deslizam entre si – elas se estendem, retraem ou movem-se para dentro e para fora. Seu funcionamento pode ser comparado ao de uma luneta.

4.2.2 Acionadores

Os *acionadores* são os dispositivos responsáveis pelo movimento das articulações do robô. Eles podem ser elétricos, hidráulicos ou pneumáticos, sendo que cada tipo possui características particulares e é indicado em diferentes situações (ROSARIO, 2010).

Os *acionadores hidráulicos* utilizam fluidos não compressíveis sob pressão, normalmente óleo, para realizar trabalho e assim movimentar as articulações. São comumente empregados em juntas prismáticas de máquinas de grande porte, pois suportam grandes cargas com um custo menor que os acionadores elétricos (ROSARIO, 2010).

Acionadores pneumáticos são muito semelhantes aos acionadores hidráulicos, com a diferença que utilizam o ar, um fluido compressível, para movimentar articulações (ROSARIO, 2010). Por este motivo, não suportam tanta carga quanto os acionadores hidráulicos. São também empregados com frequência em juntas prismáticas, mas são muito menos precisos que os acionadores hidráulicos e elétricos. Sua grande vantagem é a alta velocidade aliada a um custo relativamente baixo, sendo ideal para aplicações que exigem pouca precisão, tais como em processos que trabalham com extremos de posição.

Finalmente, os *acionadores elétricos* são dispositivos eletromecânicos que transformam energia elétrica em torque mecânico (ROSARIO, 2010). Acionadores elétricos não propiciam a velocidade dos acionadores pneumáticos ou a potência dos acionadores hidráulicos, porém possuem maior precisão que ambos. Os acionadores elétricos mais utilizados em robôs são servomotores, motores de corrente contínua (DC, do inglês *direct current*) e motores de passo. Estes acionadores são normalmente utilizados em conjunto com redutores, que diminuem a velocidade do movimento mas aumentam o torque da junta.

4.3 GRAUS DE LIBERDADE

Graus de liberdade – ou simplesmente *DoF*, do inglês *Degrees of Freedom* – é o termo utilizado para descrever a liberdade de movimento do robô no espaço bi ou tri-dimensional (CORKE, 2011). O número de graus de liberdade de um robô está relacionado com a quantidade de juntas ativas que ele possui, ou seja, juntas que são diretamente controladas por um atuador (ROSARIO, 2010). Cada junta ativa define um número de graus de liberdade e, assim, o número de graus de liberdade do robô é igual à somatória dos graus de liberdade de suas juntas (ROSARIO, 2010).

4.4 ESPAÇO DE TRABALHO

Rosario (2010) define *espaço de trabalho* como o espaço compreendido pelo conjunto de todos os pontos que um robô pode alcançar com o seu elemento terminal. Em outras palavras, trata-se do espaço onde o robô pode atuar.

O espaço de trabalho é um parâmetro importante na seleção ou no projeto de um robô, visto que ele deve ser capaz de atuar em todos os pontos de interesse do processo ao qual é aplicado. Ele depende basicamente do tamanho da estrutura mecânica e do conjunto de movimentos permitidos pelas juntas (CORKE, 2011). É importante que o espaço de trabalho do robô seja delimitado para que nada nem ninguém o transpasse, o que pode causar acidentes e danos (ROSARIO, 2010).

4.5 ESTRUTURAS TOPOLÓGICAS

A estrutura topológica se refere a forma como os elos da estrutura mecânica de um robô estão conectados (BRIOT; BONEV, 2007). A estrutura topológica determina, entre outras coisas, o espaço de trabalho do robô. O sistema controle do robô também é projetado com base em sua estrutura.

Robôs cartesianos possuem uma estrutura topológica onde os principais eixos de controle são lineares, ou seja, movem-se em linha reta. As principais vantagens apresentadas por esta estrutura são a facilidade em desenvolver-se o seu sistema de controle e sua alta precisão. Isto se deve ao fato de que, nesta estrutura, o movimento de cada eixo é independente e controlado por um único atuador.

Aplicações populares para esta estrutura topológica são as máquinas de controle numérico computacional (CNC) e impressoras 3D. Outras aplicações mais simples incluem máquinas de fresagem e de impressão (desenho), onde o elemento terminal movimenta-se no plano x-y e é levantado ou posicionado na superfície da base para criar um traços precisos. Máquinas *pick and place* e impressoras plotters também são baseadas em robôs cartesianos.

4.6 CINEMÁTICA

Cinemática é o ramo da mecânica clássica que estuda o movimento de corpos sem considerar as forças ou momentos que causam o movimento (CORKE, 2011). Na robótica, a cinemática é utilizada para descrever analiticamente o movimento de um manipulador robótico, em uma área chamada de *cinemática de robôs* (CORKE, 2011).

Segundo Corke (2011), a cinemática de robôs estuda as relações de conectividade das cadeias cinemáticas com posição, velocidade e aceleração de cada junta de um manipulador, com o objetivo de planejar e controlar os movimentos realizados pelos atuadores. A relação do movimento com as massas e forças associadas não é considerada na cinemática, sendo objeto de estudo da dinâmica de robôs.

Uma ferramenta fundamental na cinemática de robôs são as equações cinemáticas das cadeias que formam o robô (CORKE, 2011). Essas equações são utilizadas para relacionar os parâmetros das juntas com a configuração do robô e, principalmente, com a posição do elemento terminal.

Corke (2011) esclarece que a análise cinemática pode ser realizada em duas perspectivas diferentes, chamadas de *cinemática direta* e *cinemática inversa*. A *cinemática direta* utiliza as equações cinemáticas de um manipulador para calcular a posição do seu elemento terminal a partir da posição das juntas. O processo reverso, chamado de *cinemática inversa*, computa os parâmetros das juntas para que o elemento terminal alcance uma posição especificada. As dimensões do robô e suas equações cinemáticas definem o volume de espaço que pode ser alcançado pelo robô – o seu espaço de trabalho (ROSARIO, 2010).

4.7 CONTROLE DE MOVIMENTO

Rosario (2010) define os robôs como equipamentos multifuncionais reprogramáveis com grande flexibilidade de operação. Desta forma, para que um robô execute uma tarefa desejada, é necessário antes programá-lo para tanto.

Existem duas formas básicas de programar um robô, que são conhecidas como *programação online* e *programação offline* (ROSARIO, 2010). Na *programação online*, o robô é conduzido manualmente até as posições desejadas em um processo de aprendizagem de tarefas, onde o sistema de controle grava as posições das juntas e aprende o trajeto que deve realizar. Neste tipo de programação, todo o processo de aprendizagem de tarefas é realizado a partir do seu posicionamento dentro da própria célula de trabalho, com um operador manipulando-o manualmente.

Na *programação offline*, por outro lado, a programação é realizada num ambiente de modelagem, onde o usuário fornece instruções para o robô e pode visualizar resultados de simulações (ROSARIO, 2010).

Atualmente, a *programação offline* vem sendo cada vez mais utilizada como ferramenta de concepção de sistemas automatizados e programação de robôs, aumentando a flexibilidade e habilidade de utilização dos mesmos, com uma variedade ilimitada de cenários e movimentos (ROSARIO, 2010). Isso se deve principalmente à evolução de sistemas de simulação, que permitem que o sistema seja completamente simulado antes de entrar em produção.

Este tipo de programação apresenta muitas vantagens. Em primeiro lugar, o programador não precisa adentrar a área de trabalho do robô, minimizando riscos de acidentes. O tempo que o manipulador precisa ficar parado também é muito menor, visto que não é necessário realizar o processo de aprendizagem diretamente com o mesmo, mas sim em simulações. As simulações também diminuem o número de imprevistos, uma vez que todos os detalhes podem ser planejados e testados previamente, antes de o programa entrar em produção.

A programação offline é realizada através de comandos que definem o caminho (*path*) que o elemento terminal deve percorrer e as tarefas que este deve realizar. O caminho é descrito por meio de um conjunto de pontos e pelas interpolações que devem ser realizadas entre os seus pares. As tarefas, por sua vez, são definidas como comandos simples, que dependem do tipo de elemento terminal utilizado. Se este for uma garra, por exemplo, as tarefas podem se restringir a *"Abrir"* e *"Fechar"*.

Figura 26 – Programa de controle do robô SK16 escrito na linguagem Karel 2

```
PROGRAM
-- ! LANGUAGE KAREL 2
-- ! MEMORY 8192
-- ! ROBOT SK16
-- TEACHPOINT DECLARATIONS
VAR
  TP1 : POSITION
  TP2 : POSITION
  TP3 : POSITION

BEGIN
  $UTOOL=POS(-11.5265, 2.7334, 307.2149,
             0, 45, -160, ")
  $USEMAXACCEL=TRUE
  %INCLUDE TEACH POINTS#

  WITH $MOTYPE=LINEAR
    MOVE TO TP1
  -- ! ARCWELDOWN 250, 25
  WITH $MOTYPE=LINEAR
    MOVE TO TP2
  WITH $MOTYPE=LINEAR
    MOVE TO TP3
  -- ! ARCWELDOFF
END PROGRAM
```

Fonte: Adaptada de Rosario (2010)

A Figura 26 apresenta um exemplo de programa deste tipo, escrito na linguagem Karel 2, utilizada pelo controlador FANUC. Os fabricantes de robôs costumam adotar linguagens de programação próprias para programação, mas, em geral, todas possuem características semelhantes, baseadas em código G, e apresentam apenas pequenas mudanças de sintaxe e forma de estruturação (ROSARIO, 2010). No exemplo apresentado, o manipulador se move do ponto 1 até o ponto 2 e abaixa seu elemento terminal (um soldador). Após, move-se do ponto 2 até o ponto 3, executando a soldagem. Finalmente, o elemento ter-

minal é recolhido e o processo é finalizado.

Para um dado robô, os únicos parâmetros necessários para descrever os pontos do caminho são os ângulos das juntas rotativas e a posição das juntas prismáticas (CORKE, 2011). Existem, no entanto, diversas outras maneiras de definir estes pontos. A forma mais utilizada, e também mais conveniente, é especificar as coordenadas cartesianas dos pontos, ou seja, as posições do elemento terminal com relação a eixos X, Y e Z partindo de uma origem – normalmente a base do robô (CORKE, 2011). Neste caso, os ângulos das juntas são obtidos automaticamente pelo sistema de controle por meio das equações de cinemática inversa do manipulador. Além disso, o sistema realiza a interpolação e filtragem dos pares de pontos automaticamente, levando-se em consideração aspectos dinâmicos e testes de colisão (ROSARIO, 2010).

5 ROBÔ SOLUCIONADOR DE SUDOKU

Segundo Davies (2012), *visão computacional* é a área da ciência que estuda métodos de aquisição, processamento, análise e compreensão de imagens, geralmente provenientes do mundo real, para produzir informações numéricas ou simbólicas. A *visão de máquina*, por sua vez, se refere a aplicação da visão computacional para tomada de decisão em processos de automação, tais como a inspeção automática de produtos e o controle de movimento de manipuladores.

Os sistemas de visão de máquina voltados para a automação industrial formam, atualmente, um negócio multibilionário. Empresas do segmento em diferentes partes do mundo relataram grande volume de vendas no ano de 2014, confirmando as perspectivas otimistas para o setor nos próximos anos (CARROLL, 2015). Tal crescimento é impulsionado pela evolução dos dispositivos utilizados na construção destes sistemas, como sensores de aquisição de imagens e processadores, que estão cada vez mais baratos e, ao mesmo tempo, mais robustos.

Este capítulo apresenta o projeto do robô criado para solucionar jogos de Sudoku impressos em papel. Trata-se de um robô cartesiano, com dois graus de liberdade, controlado por um sistema de visão de máquina. Inicialmente, o robô captura imagens por meio de uma *webcam* e verifica se nelas existem *puzzles* Sudoku. Os jogos identificados são então interpretados e solucionados. Finalmente, o robô preenche os jogos com as soluções obtidas, escrevendo-as diretamente no papel com uma caneta, que é o elemento terminal do robô.

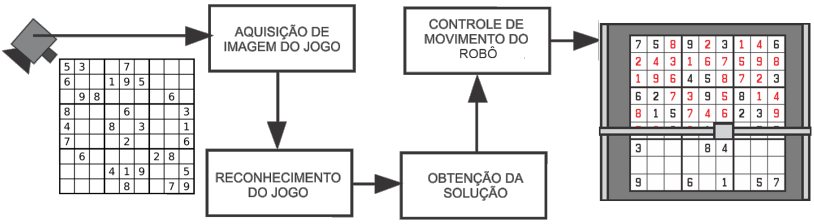
A seção seguinte apresenta uma visão geral do robô, explicando-o superficialmente e descrevendo o funcionamento esperado. Após, o projeto mecânico do robô e o sistema de controle são detalhados.

5.1 VISÃO GERAL

Este trabalho tem como objetivo último a construção de um robô capaz de solucionar jogos de Sudoku impressos em papel. Para tanto, o robô deve realizar quatro tarefas básicas: *ler*, *interpretar*, *resolver* e *escrever a solução* destes jogos.

Para desempenhar estas tarefas, propõe-se desenvolver um robô cartesiano, semelhante a máquinas industriais de controle por comando numérico (CNC), e controlá-lo por meio de um sistema de visão de máquina. O diagrama de blocos da Figura 27 ilustra a ideia.

Figura 27 – Diagrama de blocos do robô proposto



Fonte: Elaborada pelo autor

O processo se inicia com a captura de uma imagem da vista superior do espaço de trabalho do manipulador, que é realizada por meio de uma *webcam* fixada acima desta área. Após a aquisição, esta imagem é processada pelo sistema de controle, que verifica se alguma folha foi posicionada no espaço de trabalho e inicia uma busca por jogos de Sudoku presentes na mesma.

Caso algum jogo seja encontrado na imagem, o sistema o reconhece e identifica células vazias e preenchidas. Os dígitos das pistas são então processados por meio de um algoritmo de reconhecimento de caracteres (OCR) e o sistema reconstrói em memória a grade do jogo, representando-a por meio de uma matriz 9×9.

Após representar o estado inicial do sudoku por uma matriz, o sistema é capaz de solucioná-lo aplicando um algoritmo específico para esta tarefa, obtendo os dígitos que devem preencher as células vazias do jogo.

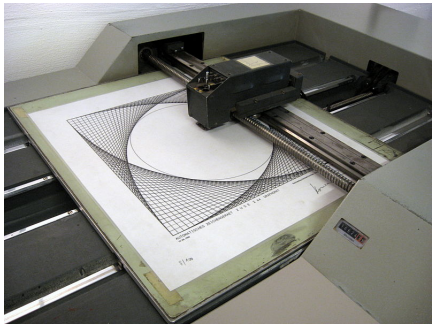
Finalmente, os números que resolvem o jogo são escritos na folha de papel. Esta tarefa envolve vários passos. Primeiro, os dígitos da solução são projetados na imagem digital, criando uma máscara. A máscara é utilizada como referência para planejar o caminho do elemento terminal, onde os pixels dos dígitos representam pontos do espaço de trabalho que devem ser riscados pela caneta. O caminho obtido com a máscara (conjunto de pontos) é descrito em coordenadas do plano cartesiano e enviado para o subsistema de controle de movimento, que utiliza as equações de cinemática inversa do robô para coordenar os atuadores e fazer com que a solução seja escrita no papel.

Os materiais e ferramentas utilizados para desenvolver o robô são apresentados no Apêndice A.

5.2 PROJETO MECÂNICO

O robô proposto baseia-se em um mecanismo planar conhecido como *plotter*. Trata-se uma estrutura cartesiana de juntas planares, capaz de movimentar o elemento terminal em um plano cartesiano x-y. A Figura 28 apresenta uma impressora que utiliza-se deste mecanismo.

Figura 28 – Impressora plotter para desenho industrial



Fonte: Campion, Wang e Hayward (2005)

As *plotters* são impressoras computadorizadas utilizadas principalmente para impressão vetorial. Sua principal aplicação é a impressão de projetos de máquinas, edificações e mapas. Tal tarefa é realizada movendo-se uma caneta ou outro instrumento similar – que funciona como elemento terminal da impressora – sobre a superfície de uma folha de papel. Isso significa que as *plotters* são dispositivos de impressão vetorial e não em linha, como impressoras comuns.

As impressoras *plotters* podem executar desenhos complexos, incluindo texto, mas o fazem de forma lenta devido ao movimento mecânico da caneta. Algumas vezes, elas são incapazes de criar uma região sólida de cor, mas podem o fazer desenhando uma série de linhas regulares.

A configuração destas impressoras foi desenvolvida com o intuito de produzir desenhos vetoriais consideravelmente grandes numa época onde a memória dos computadores era muito cara e a capacidade de processamento limitada. Elas são atualmente consideradas obsoletas e foram aos poucos substituídas por impressoras jato de tinta.

Optou-se pelo emprego desta estrutura no robô solucionador de Sudoku pelo fato de a mesma apresentar características desejáveis para a aplicação final: rigidez, precisão e velocidade (BRIOT; BONEV, 2007).

Outra justificativa para o seu emprego é o fato de que pretende-se construir o robô exclusivamente com peças e acionadores dos kits *LEGO Mindstorms EV3* e *LEGO Technic*, inviabilizando a utilização de estruturas como braços mecânicos devido à folgas e pouca estabilidade das estruturas construídas com LEGO.

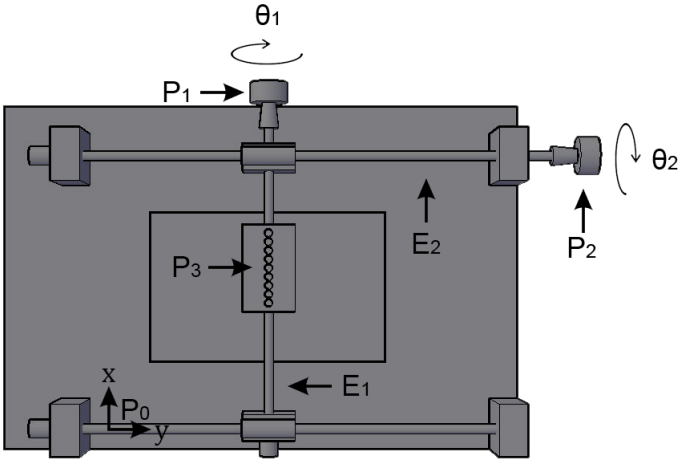
5.2.1 Cinemática

Esta seção apresenta em detalhes a configuração cinemática de uma *plotter*. Seu entendimento é essencial para implementação do sistema de controle do robô.

5.2.1.1 Cinemática Direta

O modelo de cinemática utilizado para o problema direto de uma *plotter* é apresentado na Figura 29. O elemento terminal está localizado no ponto $P3$ e se move em um plano com dois graus de liberdade em relação à base. Atuadores encontram-se fixados nos pontos $P1$ e $P2$. A posição do elemento terminal é determinada pela posição dos ângulos destas juntas, θ_1 e θ_2. A área compreendida pelo retângulo ao centro ilustra o espaço de trabalho útil desejado.

Figura 29 – Modelo de cinemática direta da plotter



Fonte: Elaborada pelo autor

O problema da cinemática direta consiste, portanto, em encontrar a posição do ponto P_3 a partir dos ângulos θ_1 e θ_2 das juntas ativas. O sistema de referências tomado é tal que eixo z passa por P_0 e o plano x-y é o mesmo do espaço de trabalho.

É possível observar que o ponto P_3 encontra-se na intersecção dos elos E_1 e E_2. O elo E_1 move-se pelo eixo y por meio da rotação angular do motor situado em P_2. Assim, a posição do mesmo tem relação linear com o ângulo θ_2. Em termos de P_3, considerando que $\theta_2 = 0$ quando $y = 0$, obtém-se a relação apresentada na Equação (5.1) para determinar sua posição em y, sendo a uma constante real dependente do dimensionamento mecânico da estrutura.

$$y_3 = a\theta_2 \tag{5.1}$$

Raciocínio semelhante pode ser aplicado ao eixo x de P_3, que se move nesta direção através da rotação angular do motor situado em P_1. Em termos de P_3, considerando $\theta_1 = 0$ quando $x = 0$, obtém-se a relação apresentada na Equação (5.2) para determinar sua posição em x, sendo b uma constante real dependente do dimensionamento mecânico da estrutura.

$$x_3 = b\theta_1 \tag{5.2}$$

A posição do elemento terminal $P_3 = (x_3, y_3)$ é dada, portanto, pela Equação (5.3).

$$(x_3, y_3) = (b\theta_1, a\theta_2) \tag{5.3}$$

5.2.1.2 Cinemática Inversa

O problema da cinemática inversa para a *plotter* consiste em encontrar o valor dos ângulos θ_1 e θ_2 a partir da localização do ponto P_3. A solução deste problema é importante para determinar como as juntas devem ser posicionadas para que o elemento terminal alcance um dado ponto do espaço de trabalho.

Por meio de simples observação da Equação (5.3), é possível estabelecer as relações apresentadas na Equação (5.4), obtendo o modelo de cinemática inversa do problema.

$$\theta_1 = \frac{x_3}{b}, \qquad \theta_2 = \frac{y_3}{a} \tag{5.4}$$

5.2.2 Dimensionamento

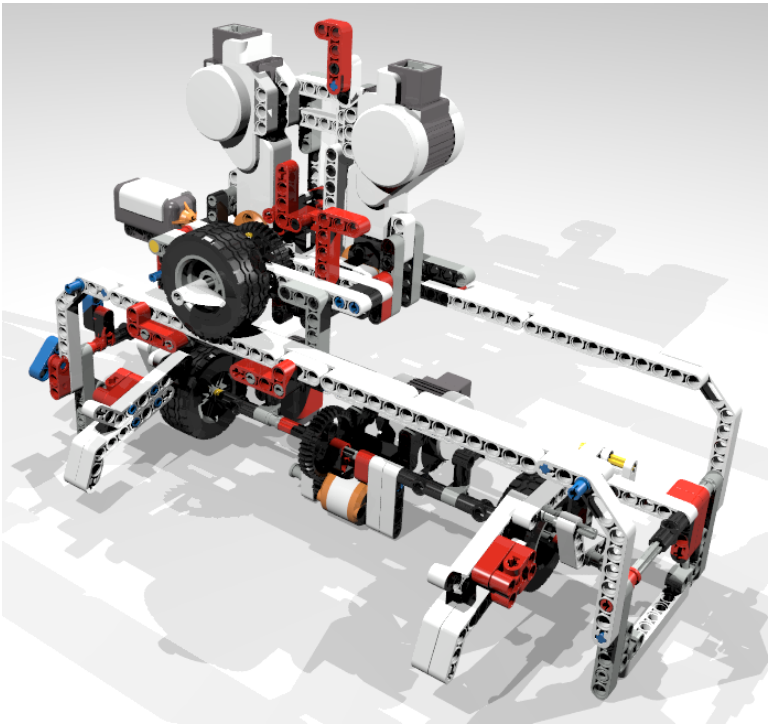
O robô foi dimensionado de forma que o espaço de trabalho cobrisse uma área mínima de interesse, chamada de *espaço de trabalho útil*. As dimensões deste espaço foram definidas como sendo iguais às de uma folha A4, ou seja, com 210 mm de largura e 297 mm de altura (210 mm $\times$ 297 mm). Como a carga a ser suportada pela estrutura consiste apenas no peso de algumas peças de LEGO, nenhuma carga atuante sobre a estrutura foi considerada durante o dimensionamento.

5.2.3 Modelagem

O robô foi projetado[1] exclusivamente com peças dos kits *LEGO Mindstorms* e *LEGO Technic*. O software *MLCad* foi utilizado para realizar o desenho, com apoio da biblioteca *LDraw*. A Figura 30 apresenta o modelo 3D do projeto, renderizado com a ferramenta *PoV Ray*.

[1] O projeto completo do robô, assim como a lista de peças utilizadas e instruções de montagem, encontra-se disponível em *http://thiagozf.github.io/sudokurobot*.

Figura 30 – Modelo 3D do robô solucionador Sudoku



Fonte: Elaborada pelo autor

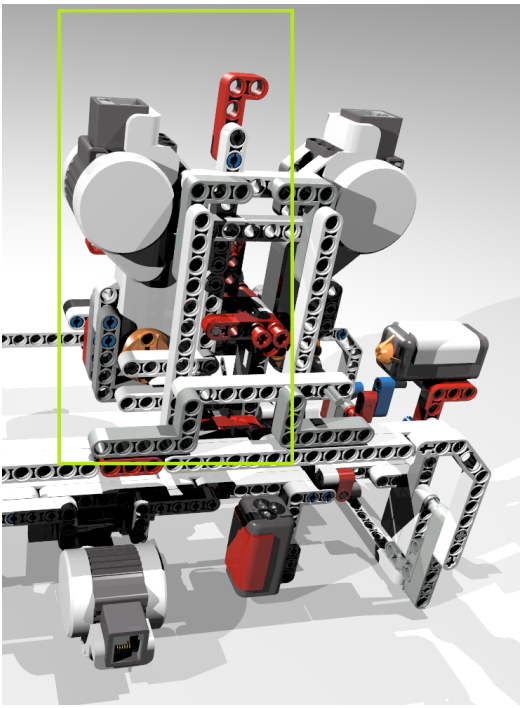
Os acionadores usados para movimentar as juntas ativas da estrutura foram servomotores do kit *Mindstorms*. Estes servos rodam com velocidades angulares que variam de 160 até 170 rpm e possuem torque de 20 N cm (torque inicial de 40 N cm). Eles também possuem um sensor interno (tacômetro) com 1° de resolução, permitindo um controle preciso dos movimentos.

O elo E_2, responsável por controlar o movimento do elemento terminal ao longo do eixo y, foi substituído por uma base que movimenta a folha de papel ao invés de movimentar o elo E_1 – e, por consequência, o elemento terminal. Esta modificação apresenta algumas vantagens. A primeira delas é a redução do tamanho físico do robô, que pode ser consideravelmente reduzido diminuindo o eixo y. O torque necessário para movimentar a folha de papel também é consideravelmente menor do que o torque para movimentar todo o elo E_1. A alteração, portanto, também reduz a carga sob a qual está sujeito o motor da junta P_2. Isto

pode se contribuir para o aumento de vida útil do motor.

As constantes a e b da Equação (5.4), utilizadas para obter os ângulos dos motores para posicionar o elemento terminal, são determinadas experimentalmente durante a fase de calibração do dispositivo construído.

Figura 31 – Mecanismo para posicionamento do elemento terminal



Fonte: Elaborada pelo autor

Uma vez que o espaço de trabalho do robô se restringe a um plano, o controle de posição do elemento terminal no eixo Z não foi considerado. Existe, no entanto, um mecanismo capaz de retirar e colocar a caneta em contato com o papel. Este mecanismo funciona como uma alavanca que inclina a caneta para baixo, fazendo com que a mesma risque o papel durante os movimentos subsequentes, ou para cima, permitindo que o elemento terminal se mova sem que o papel seja riscado. Este mecanismo é controlado também por um servomotor. A Figura 31 mostra detalhes do mecanismo para controle do elemento

terminal.

5.3 SISTEMA DE CONTROLE

O *sistema de controle* é o software responsável por *ler*, *interpretar* e *resolver* os jogos de sudoku fornecidos como entrada para o robô. Ele é também responsável por *coordenar os atuadores*, de modo que a escrever as soluções dos jogos no papel, controlando posição, velocidade e aceleração dos acionadores que movimentam o elemento terminal.

O *brick* EV3 é o único dispositivo utilizado no controle do robô. O sistema proposto funciona como uma aplicação do sistema operacional *ev3dev*, valendo-se dos drivers nativos desta distribuição para controlar os servomotores LEGO e a *webcam*. O sistema[2] foi projetado com o *MATLAB* e sua *toolbox* de processamento de imagens, permitindo a criação rápida de um protótipo para validação das técnicas empregadas no reconhecimento dos jogos de Sudoku. Após, o sistema foi desenvolvido utilizando a linguagem *Python*, a qual é suportada oficialmente pelo *ev3dev*.

Esta seção apresenta o sistema desenvolvido para controle do robô e os resultados obtidos com o mesmo. O sistema foi dividido em 4 subsistemas, sendo cada um responsável por uma tarefa diferente: *aquisição de imagens*, *reconhecimento dos jogos*, *resolução dos jogos* e *escrita das soluções*. Cada um deles é detalhado nas seções seguintes.

5.3.1 Aquisição de Imagem

A primeira tarefa executada pelo sistema de controle é a captura de uma imagem do jogo a ser resolvido, fotografando o espaço de trabalho do robô. A aquisição da imagem é realizada utilizando a *webcam* Logitech C270. A câmera foi fixada acima do espaço de trabalho, obtendo imagens de 640×480 pixels dos jogos com distorção de perspectiva mínima. A Figura 32 apresenta a imagem de um jogo obtida nesta condição.

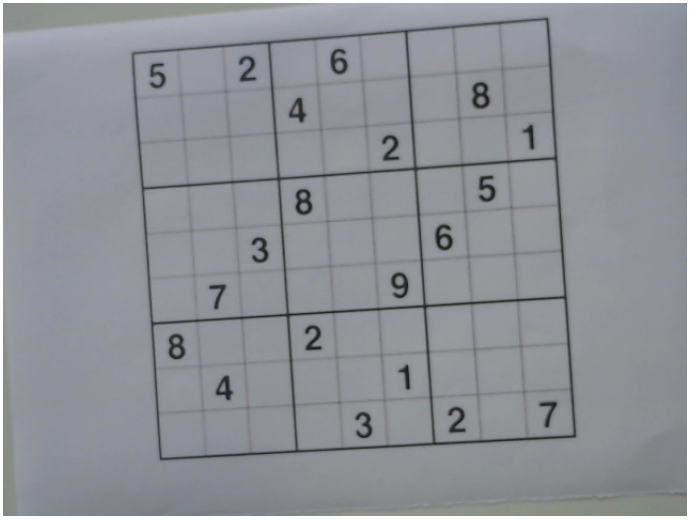
A *webcam* é conectada ao *brick* EV3 pela porta USB. A comunicação entre os dispositivos é realizada utilizando o protocolo *UVC*, suportado pela *webcam* e pelo sistema operacional.

Após obter a imagem do jogo de Sudoku, o sistema executa uma

[2] O código-fonte do sistema e do protótipo escrito em MATLAB podem ser obtidos em *http://thiagozf.github.io/sudokurobot*.

série de processamentos sobre a mesma com o objetivo de reconhecer o jogo escrito na folha.

Figura 32 – Exemplo de imagem de entrada do sistema



Fonte: Elaborada pelo autor

5.3.2 Reconhecimento do Jogo

O reconhecimento do jogo é a tarefa mais complexa realizada pelo sistema de controle. O processo é dividido em diversas etapas. Primeiro, a imagem colorida obtida durante a aquisição é convertida em uma imagem binária. Após, uma possível inclinação da grade na imagem é corrigida e suas linhas são detectadas. As áreas da imagem que correspondem a cada uma das células da grade são então determinadas localizando os pontos de intersecção das linhas detectadas na etapa anterior. Finalmente, as áreas da imagem correspondente às células são processadas por um algoritmo de reconhecimento óptico de caracteres (OCR), identificando as pistas do sudoku.

Cada etapa do processo de reconhecimento do jogo é detalhada nas subseções seguintes. Ao final, resultados experimentais obtidos com o sistema de reconhecimento são apresentados.

5.3.2.1 Premissas

Antes de desenvolver o sistema de reconhecimento, algumas premissas foram estabelecidas. A primeira é que os sudokus fornecidos para reconhecimento são jogos válidos, ou seja, não violam nenhuma regra do Sudoku. A segunda premissa é que um jogo a ser reconhecido estará localizado aproximadamente no centro da imagem. Por fim, os jogos deverão apresentar rotação máxima de $\pm 45°$ em relação às bordas horizontais da imagem, ou seja, jogos muito rotacionados – de cabeça para baixo em relação à câmera, por exemplo – não serão reconhecidos pelo sistema.

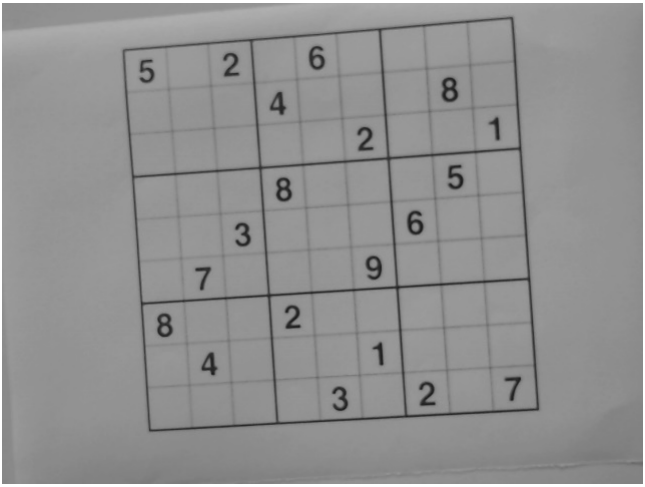
5.3.2.2 Binarização

A primeira operação executada pelo sistema de reconhecimento é a binarização, convertendo a imagem colorida obtida durante a aquisição em uma imagem binária. O objetivo da binarização é segmentar os objetos de interesse na imagem, que são as linhas da grade e as pistas. Todas as operações subsequentes necessárias para reconhecer o jogo, em especial o algoritmo de OCR, são facilitadas trabalhando-se com uma imagem binária.

Antes de realizar a binarização, é necessário obter a representação em escala de cinza da imagem. A Equação (3.3) é aplicada sobre a imagem de entrada. O resultado é apresentado na Figura 33.

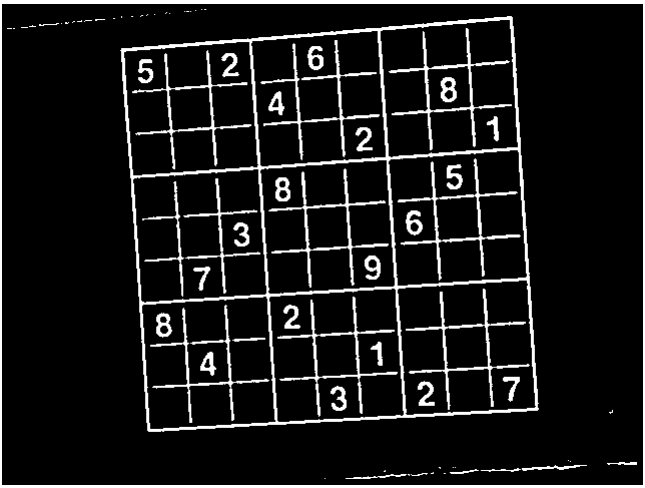
A binarização da imagem é finalmente realizada por meio de uma operação de thresholding. O método de Sauvola, um dos mais eficientes métodos de thresholding adaptativo local, é utilizado (SAUVOLA; PIETIKäINEN, 2000). A Equação (3.8) é aplicada sobre a imagem em escala de cinza, usando uma janela deslizante de tamanho $M \times M = 11 \times 11$ pixels e uma tolerância de $C = 4$. A Figura 34 apresenta o resultado desta operação sobre a imagem de exemplo apresentada na Figura 33.

Figura 33 – Imagem de entrada em escala de cinza



Fonte: Elaborada pelo autor

Figura 34 – Imagem de entrada após binarização



Fonte: Elaborada pelo autor

A razão principal para utilizar um método adaptativo local para realizar a binarização é a robustez apresentada por esta classe de al-

goritmos em situações onde existem gradientes de iluminação sobre a imagem, as quais podem ser causadas, por exemplo, pela sombra de uma pessoa próxima ao robô. Um método de thresholding global nesta situação poderia remover detalhes importantes da grade e dos dígitos, impossibilitando o reconhecimento dos mesmos. Outro motivo para esta decisão foi a pesquisa realizada por Du et al. (2013), apontando que esta classe de métodos apresenta melhores resultados em aplicações de análise de documentos.

A implementação do método de Sauvola foi realizada utilizando a técnica de *imagem integral*, reduzindo significativamente o tempo de processamento necessário para obter as médias das janelas e, consequentemente, o tempo para completar a operação.

5.3.2.3 Correção de Inclinação

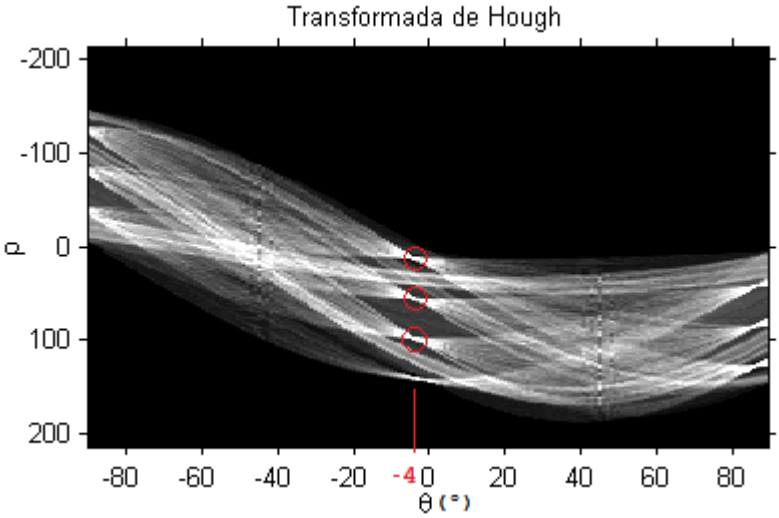
Apesar de o posicionamento da câmera contribuir para que não ocorra distorção de perspectiva dos jogos nas imagens de entrada, é quase certo que este apresente inclinação. Diz-se que ocorre inclinação quando o conteúdo do documento na imagem estiver rotacionado em relação às bordas da imagem. A imagem de entrada que serve de exemplo nesta seção (Figura 32), por exemplo, possui uma inclinação de cerca de $-4°$.

A inclinação pode representar um grande problema para o reconhecimento de caracteres caso não seja devidamente corrigida antes da execução do algoritmo de OCR. A correção da inclinação do conteúdo não é uma tarefa simples em alguns tipos de documentos. Felizmente, as características dos jogos de Sudoku facilitam este trabalho.

A grade de Sudoku é composta por um total de 10 linhas horizontais e 10 linhas verticais, sendo todas retas. Estas linhas apresentam boa definição nas imagens capturadas pelo processo de aquisição e, portanto, podem ser utilizadas como referência na determinação do ângulo de inclinação do jogo. As linhas da grade são localizadas pelo sistema aplicando-se a *transformada de Hough* à imagem binária obtida na etapa anterior. Como resultado desta operação, são obtidas as linhas em sua forma polar, ou seja, descritas em termos da distância até a origem e do seu ângulo de inclinação em relação a esta. O ângulo de inclinação deve ser igual, ou muito próximo, para todas as linhas na mesma orientação (horizontal ou vertical). Para poupar tempo de processamento, o sistema realiza a operação apenas sobre um quadrado de 150×150 no centro da imagem. A Figura 35 apresenta o resultado

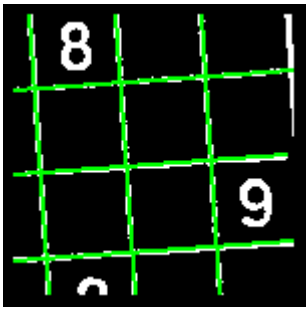
da transformada de Hough sobre a parte central da imagem, destacando os pontos de máximo que representam as linhas horizontais da grade. A Figura 36 destaca as linhas obtidas com a transformada sobre a imagem binária.

Figura 35 – Transformada de Hough sobre a imagem



Fonte: Elaborada pelo autor

Figura 36 – Linhas obtidas com a transformada de Hough



Fonte: Elaborada pelo autor

A correção de inclinação do jogo é realizada rotacionando a imagem na mesma proporção do ângulo das linhas, obtido pela transfor-

mada de Hough, no sentido contrário. A rotação de uma imagem $I(x, y)$ é realizada multiplicando um operador matricial de rotação pelas coordenadas (x, y) de cada pixel da mesma, descritos por um vetor coluna na forma $[x\ y]^\top$. O resultado é a nova posição do pixel (x', y'), também na forma de um vetor coluna $[x'\ y']^\top$ (GONZALEZ; WOODS, 2008). A operação é descrita pela Equação (5.5). É importante observar que a imagem rotacionada irá ocupar pontos fora das dimensões originais da imagem e, portanto, a imagem resultante do processo de rotação terá dimensões diferentes. Pontos fora da área rotacionada da imagem são preenchidos com '0'.

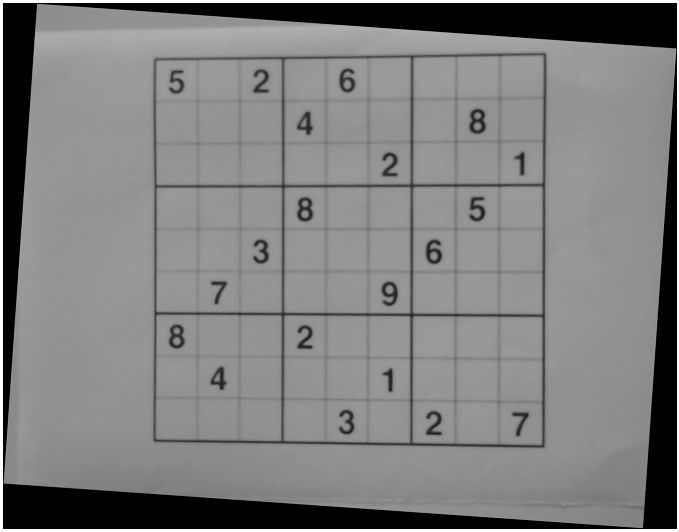
$$\begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x' \\ y' \end{bmatrix} \tag{5.5}$$

Outra observação relevante é que, após aplicar o operador de rotação, nem todos os pixels da área rotacionada são preenchidos, deixando espaços dentro da área da imagem original. Para preencher estes espaços, utiliza-se algum método de interpolação (GONZALEZ; WOODS, 2008). As interpolações *por vizinho mais próximo*, *bilinear* e *bicúbica* são as mais utilizadas em processamento de imagens. Como mostra Gonzalez e Woods (2008), imagens reamostradas com a interpolação bicúbica apresentam um resultado de melhor visualização, incorrendo em menos erros de interpolação. A interpolação bicúbica é realizada em um pixel tomando uma média ponderada da intensidade de sua vizinhança quadrada 4×4, conforme a Equação (5.6). Os pesos a_{ij} são os coeficientes de splines bicúbicos entre os pixels de linhas e colunas desta vizinhança. Franco (2006) apresenta o processo para calcular os coeficientes.

$$g(x, y) = \sum_{i=0}^{3} \sum_{j=0}^{3} a_{ij} x^i y^j \tag{5.6}$$

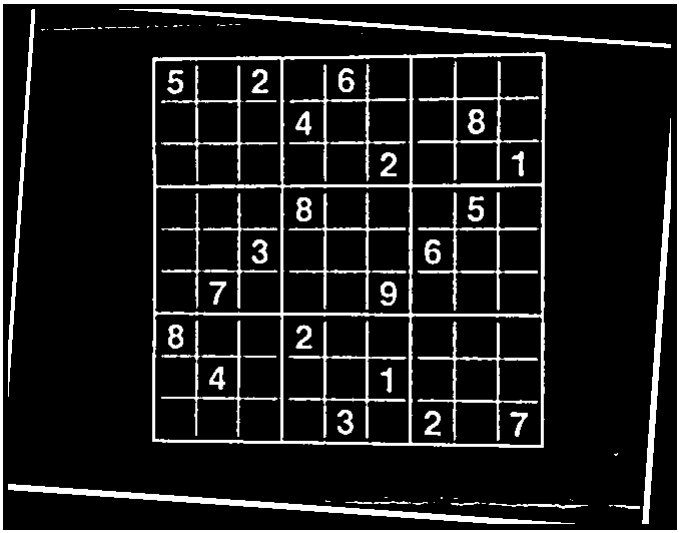
A interpolação bicúbica é utilizada pelo sistema para realizar a interpolação dos pixels na imagem transformada pela rotação da imagem original. A Figura 37 apresenta o resultado das operações descritas sobre a imagem em escala de cinza do jogo.

Figura 37 – Correção de inclinação da imagem de entrada



Fonte: Elaborada pelo autor

Figura 38 – Imagem binária com inclinação corrigida



Fonte: Elaborada pelo autor

A imagem é novamente binarizada após a correção de inclinação na imagem em escala de cinza. O processo não é aplicado diretamente na imagem binária obtida anteriormente para conservar a integridade dos dígitos, que seriam deformados pela interpolação. A Figura 38 mostra a imagem binária com inclinação corrigida.

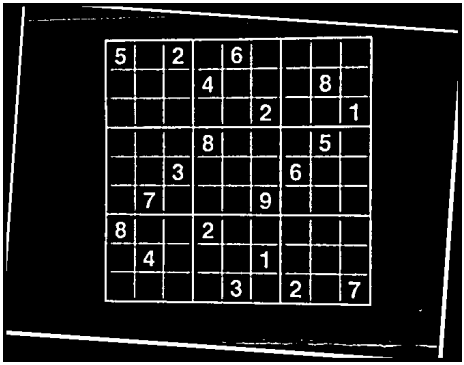
5.3.2.4 Localização da Grade

O próximo passo para reconhecer os jogos é segmentar a grade do restante da imagem. Isto é feito para eliminar ruídos que possam prejudicar as etapas seguintes.

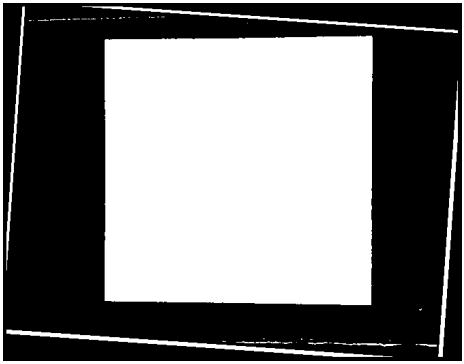
O processo para localização da grade se inicia com uma operação morfológica de fechamento sobre a imagem binária da Figura 38. O resultado esperado por esta operação é a obtenção de uma nova imagem, onde a grade do jogo se encontra completamente preenchida com pixels de foreground (valor '1').

As regiões da imagem após o fechamento são então analisadas uma a uma. Espera-se que a região de maior área seja a região da grade, obtida com o fechamento realizado anteriormente. Como esta é área de interesse na imagem, todas as demais regiões são excluídas da imagem. A Figura 39 ilustra o procedimento.

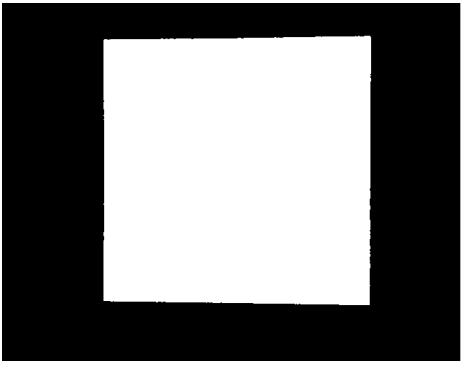
Figura 39 – Operações para detecção da grade do jogo



(a) Imagem binária original



(b) Resultado da operação de fechamento

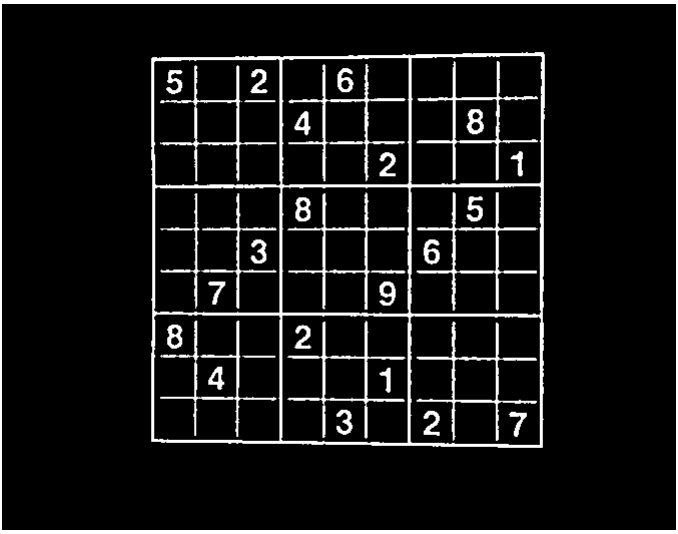


(c) Segmentação da região de maior área

Fonte: Elaborada pelo autor

A segmentação da grade na imagem binária é então realizada por meio de uma operação lógica *AND* pixel a pixel entre ela e a imagem resultante das últimas operações (Figura 39). A grade do jogo segmentada é apresentada na Figura 40.

Figura 40 – Localização da grade na imagem de entrada



Fonte: Elaborada pelo autor

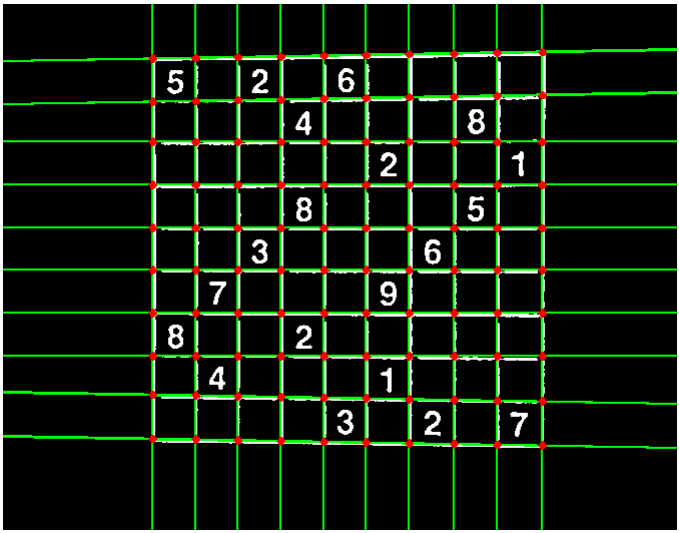
5.3.2.5 Localização das Células

O próximo passo para completar o reconhecimento do jogo é localizar as células da grade na imagem de entrada. A localização das células consiste em determinar os quatro vértices dos quadrados que as delimitam.

Esta tarefa é realizada pelo sistema obtendo os pontos de intersecção das linhas compreendidas dentro da grade. As linhas são obtidas por meio da transformada de Hough aplicada à imagem da Figura 40. Os pontos de intersecção entre elas são então calculados por meio da resolução de sistemas de equações lineares compostos por pares de linhas, uma de orientação vertical e outra de orientação horizontal. O método numérico utilizado para resolução destes sistemas foi a decomposição LU, detalhado por Franco (2006). A Figura 41 apresenta os pontos de intersecção detectados, destacados em vermelho. As linhas detectadas

pela transformada de Hough são destacadas em verde.

Figura 41 – Localização das células na imagem de entrada



Fonte: Elaborada pelo autor

Agora que as coordenadas de todos os pontos da grade são conhecidas, a imagem do jogo é dividida nos 81 quadrados delimitados pelos pontos, e cada um é classificado como uma pista ou como uma célula vazia. A classificação é realizada segmentando os regiões conectados dentro do quadrado. Caso nenhuma região de foreground exista no quadrado, ou nenhum dos componentes obtidos possua uma quantidade considerável de pixels para ser um dígito, a célula é considerada vazia. Se regiões com uma quantidade mínima de pixels de foreground forem encontradas, aquela com maiores chances de ser um número é processada pelo algoritmo de OCR. Esta seleção é baseada na posição onde a região foi encontrada e no tamanho da mesma. A Figura 42 mostra a imagem ampliada de um dos quadrados obtidos pelo sistema, destacando as regiões de foreground conectadas e a região do dígito segmentada após o processo. Para evitar que pixels do quadrado que delimita a célula estejam presentes durante a busca pelo dígito, regiões conectadas aos limites da célula são removidas. Uma margem de 10% para altura e largura também é considerada.

Figura 42 – Célula do jogo segmentada da imagem de entrada

(a) Célula (b) Remoção de bordas (c) Enquadramento

Fonte: Elaborada pelo autor

5.3.2.6 Reconhecimento de Pistas

O reconhecimento das pistas é a última etapa necessária para completar o reconhecimento do jogo. Nesta etapa, as células que contém pistas são processadas por um algoritmo de OCR, responsável por identificar quais são os números presente na imagens fornecidas como entrada.

Várias técnicas podem ser empregadas para reconhecer caracteres. Algumas delas são extremamente robustas, mas exigem um tempo alto de execução e capacidade de processamento. Outras são mais simples e apresentam uma taxa de acertos menor, porém podem ser executadas sem grande esforço computacional. A técnica selecionada depende diretamente da aplicação a qual ela se destina (BOROVIKOV; LANE, 2004).

O *template matching* foi implementado como método de reconhecimento das pistas no sistema de controle. Esta escolha se justifica pelo fato de que apenas dígitos numéricos de '1' até '9' são possíveis, ou seja, o algoritmo deve selecionar o número correto em um total de apenas 9 templates diferentes. Com um conjunto reduzido de possibilidades, este método tende a apresentar uma boa taxa de acertos e um tempo de execução pequeno. A função de similaridade utilizada foi a *distância de Manhattan*, dada pela Equação (3.12).

Os templates foram obtidos manualmente, tomando a média de intensidade de cada pixel em imagens binárias dos dígitos pertencentes a um conjunto de treinamento[3]. O conjunto de treinamento foi composto por 16 amostras de cada número, retiradas de fotos de jo-

[3] O conjunto de treinamento utilizado encontra-se disponível em *http://thiagozf.github.io/sudokurobot*.

gos capturadas nas mesmas condições em que as imagens de entrada do sistema serão. Todas as amostras foram normalizadas para as dimensões de 10×15 pixels, tamanho definido de modo empírico para os templates. A Figura 43 apresenta os templates obtidos como resultado do processo de treinamento.

Figura 43 – Templates dos dígitos para OCR

Fonte: Elaborada pelo autor

O reconhecimento foi aperfeiçoado com heurística para identificação do número '1'. Este dígito possui a peculiaridade de possuir uma largura muito menor que a dos demais números, mantendo a mesma altura. Assim, todo dígito enquadrado com uma largura menor do que 40% da sua altura é considerado '1' pelo sistema. A relação de proporcionalidade entre altura e largura foi obtida empiricamente, medindo as amostras do conjunto de treinamento. Esta relação é útil pois, quando normalizado, o dígito '1' apresenta grande semelhança com o dígito '7'. Com este aperfeiçoamento, números da grade de Sudoku deixam de ser comparados com o template do número '1', que passa a ser identificado apenas pela relação estabelecida.

Os dígitos reconhecidos são finalmente dispostos em uma matriz numérica de dimensões 9×9, representando a grade de Sudoku. Células vazias são representadas pelo número 0. A posição de um número na matriz é determinada pela localização de sua célula na imagem, obtida antes da etapa de reconhecimento. Nesta representação, o jogo está pronto para ser solucionado por um dos algoritmos apresentados na Seção 2.4.

5.3.2.7 Pós-processamento

Após reconhecer os dígitos e obter a representação do jogo na forma de uma matriz numérica, algumas verificações são executadas

pelo sistema em busca de erros no reconhecimento. O sistema avalia se ocorre repetição de algum número nos grupos do jogo (linhas, colunas ou regiões). Nestes casos, as células que possuírem números repetidos reconhecidos com as maiores distâncias são reprocessadas, e os próximos números na fila de probabilidade são tomados como os valores para estas células.

5.3.2.8 Resultados

Um conjunto de testes[4] composto por 30 imagens de boa qualidade, capturadas nas mesmas condições em que serão as imagens de entrada do sistema, foi processado pelo sistema. O intuito do experimento foi medir a taxa de acertos do reconhecedor proposto, verificando se ele é robusto o suficiente para utilização na aplicação final.

O sistema desenvolvido apresentou uma taxa de acertos de 100%, reconhecendo corretamente um total de 602 pistas presentes nos jogos do conjunto de testes. Nenhum jogo foi corrigido durante a etapa de pós-processamento. Este aperfeiçoamento, no entanto, foi mantido para aumentar a confiança do sistema. Todos os jogos foram reconhecidos em tempos inferiores a 100 ms, o que é considerado aceitável para o robô.

5.3.3 Solução do Jogo

Depois de reconhecer o jogo em uma imagem de entrada, o sistema precisa solucioná-lo. Existem diversos algoritmos capazes de realizar esta tarefa, cada um com uma abordagem diferente do problema. A Seção 2.4 apresentou, em versões simplificadas, três dos algoritmos mais utilizados em solucionadores determinísticos do estado da arte: *Backtracking*, *Dancing Links* e *Rule-based* (BERGGREN; NILSSON, 2012).

O algoritmo de Backtracking aperfeiçoado com as técnicas de Minimum Remaining Values (MRC) e Forward Checking (FC) foi selecionado para utilização no sistema. A escolha foi realizada com base em experimentos conduzidos, onde este algoritmo se mostrou suficientemente eficaz para a aplicação final.

[4] O conjunto de testes utilizado encontra-se disponível em *http://thiagozf.github.io/sudokurobot*.

5.3.4 Escrita da Solução

Após ler, reconhecer e resolver o jogo, o robô deve ainda escrever sua solução no papel. A trajetória do elemento terminal no espaço de trabalho determina o que será escrito na folha. Desta forma, o problema de escrita da solução se resume a projetar a trajetória e controlar os acionadores para executá-la. Este processo é semelhante ao executado por máquinas industriais CNC, com a diferença que o robô solucionador de Sudoku deve projetar a sua trajetória automaticamente.

As próximas seções apresentam os procedimentos adotados pelo sistema para estabelecer o caminho a ser percorrido pelo elemento terminal e para controlar o movimento do robô de forma que ele execute a trajetória planejada.

5.3.4.1 Cálculo da Trajetória

O movimento de robôs é, em geral, determinado por meio de comandos que definem a trajetória que o elemento terminal deve percorrer no espaço de trabalho (ROSARIO, 2010). Este caminho é normalmente descrito por um conjunto de pontos em coordenadas cartesianas e pelas interpolações que devem ser realizadas entre os seus pares.

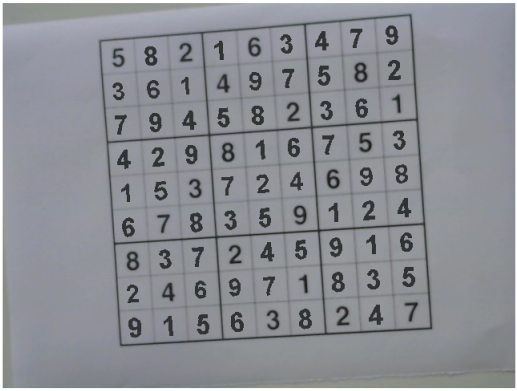
O robô solucionador de Sudoku deve estabelecer automaticamente a trajetória para escrita da solução de um jogo. Para tanto, o sistema de controle se vale de algumas técnicas de visão de máquina. A ideia consiste em projetar os dígitos que solucionam o jogo sobre a imagem de entrada, criando uma máscara que represente virtualmente o caminho a realizar.

Templates de imagens binárias dos 9 dígitos possíveis são armazenados no sistema de controle. As coordenadas das células vazias, obtidas em etapas anteriores, são utilizadas como referência para posicionar e redimensionar os templates dos números que devem ser inseridos nestas células para solucionar o jogo. Após posicionar todos os números da solução na máscara, esta é rotacionada para considerar a inclinação na imagem de entrada original, a qual foi corrigida antes de localizar das células. A Figura 44 apresenta a máscara e a projeção da mesma sobre o jogo de exemplo.

Figura 44 – Solução do sudoku projetada sobre a imagem original

	8		1		3	4	7	9
3	6	1		9	7	5		2
7	9	4	5	8		3	6	
4	2	9		1	6	7		3
1	5		7	2	4		9	8
6		8	3	5		1	2	4
	3	7		4	5	9	1	6
2		6	9	7		8	3	5
9	1	5	6		8		4	

(a) Máscara da solução



(b) Solução projetada na imagem

A trajetória do elemento terminal é por fim calculada transformando a localização dos pixels de foreground da máscara em coordenadas cartesianas do espaço de trabalho do robô. Esta transformação resulta em um conjunto de pontos que descreve o caminho a ser realizado pelo robô. A relação entre coordenadas reais do espaço de trabalho com os pixels nas imagens de entrada é obtida calibrando o sistema de controle.

5.3.4.2 Controle do Movimento

Depois de calcular a sequência de pontos que descreve a trajetória a ser seguida para escrita da solução no papel, a última operação executada pelo sistema é o controle de movimento do robô, de forma que o elemento terminal percorra o caminho obtido.

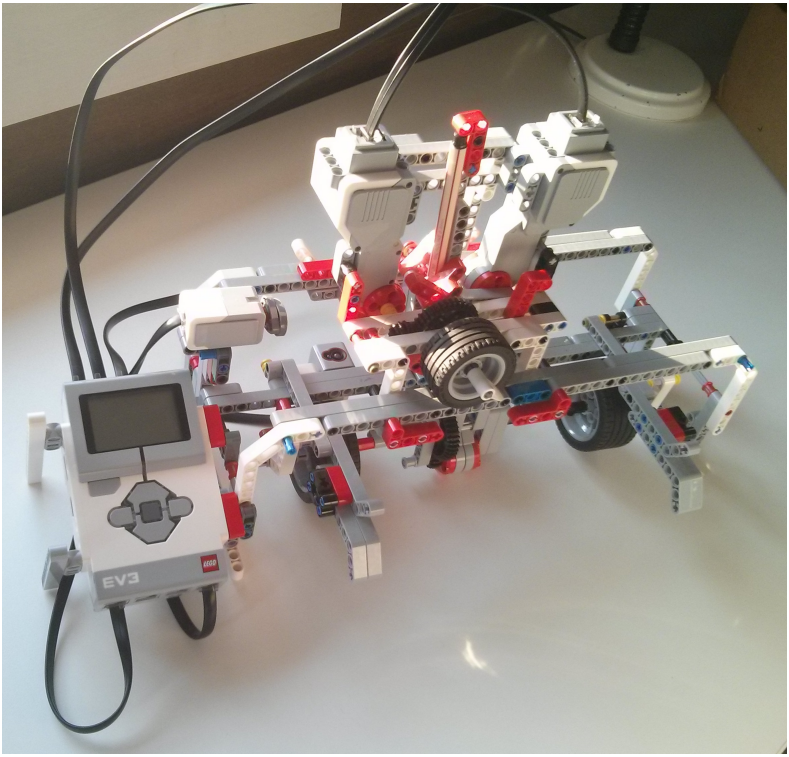
O movimento é realizado controlando as posições das juntas ativas da estrutura mecânica, apresentada em detalhes na Seção 5.2. O sistema itera sobre cada ponto da trajetória e, baseando-se nas relações de cinemática inversa do robô, sintetizadas pela Equação (5.4), calcula os ângulos das juntas θ_1 e θ_2 para que o elemento terminal alcance este ponto (P_3) no espaço de trabalho. Cada junta ativa é movimentada por um servomotor.

A escrita de cada dígito da solução é realizada por um movimento contínuo das juntas. O sistema posiciona o elemento terminal recolhido – ou seja, sem que esteja em contato com o papel – no primeiro ponto da trajetória que descreve o movimento necessário para escrita de um dígito numa célula vazia. Após, elemento terminal é posicionado no plano do espaço de trabalho – ou seja, entra em contato com a folha –, e o restante da trajetória é executada continuamente. Ao finalizar a escrita de um dígito, o elemento terminal é recolhido e o processo se repete até que toda a solução tenha sido escrita.

5.4 RESULTADOS

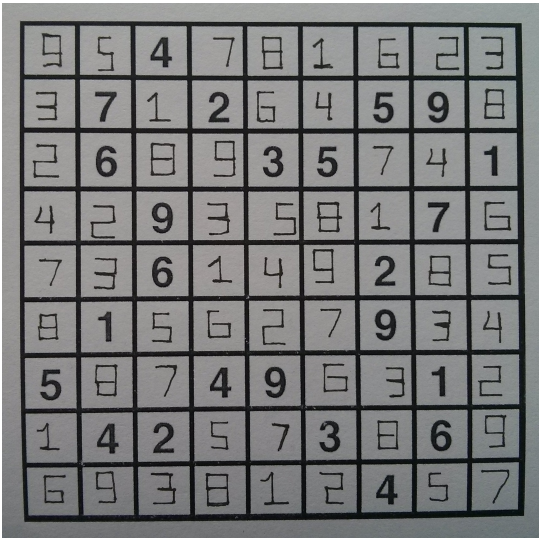
Após desenvolver o projeto e o sistema de visão de máquina, o robô foi montado. A Figura 45 apresenta uma foto do mesmo. Diversos testes foram conduzidos com o intuito de validar o funcionamento do robô construído. Em geral, os resultados mostraram-se satisfatórios. A Figura 46 apresenta um jogo de Sudoku resolvido com sucesso pelo robô.

Figura 45 – Robô solucionador de Sudoku



Fonte: Elaborada pelo autor

Figura 46 – Jogo solucionado com sucesso pelo robô



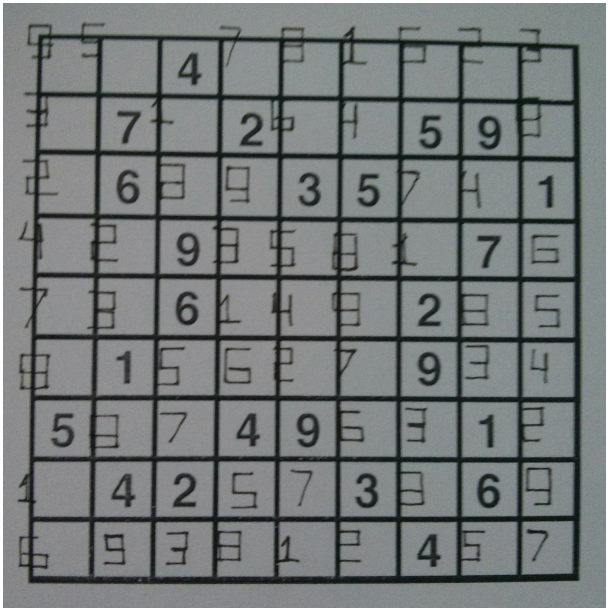
Fonte: Elaborada pelo autor

Nota-se, entretanto, que em algumas ocasiões ocorrem pequenas falhas em determinar a localização física do jogo e de suas células na folha de papel. A Figura 47 ilustra o exemplo de uma solução obtida com pequenas falhas na escrita. Dois fatores contribuem para que esta situação aconteça:

- A baixa rigidez de construções montadas com peças de LEGO em relação à rigidez ideal para um dispositivo mecânico de precisão como o robô construído, a qual faz com que os movimentos do elemento terminal não sejam tão precisos quanto o esperado;

- Baixa estabilidade da estrutura de suporte para fixação da *web-cam*, responsável por obter as imagens dos jogos e fornecê-las como entrada ao robô, fazendo com que pequenas trepidações no ambiente durante a obtenção da imagem prejudiquem a localização do jogo pelo sistema de visão.

Figura 47 – Jogo solucionado pelo robô com pequenas falhas



Fonte: Elaborada pelo autor

O tempo médio para processar cada imagem de entrada fornecida ao sistema foi de aproximadamente 40 segundos. Cerca de 90% deste tempo é utilizado para processar as etapas de localização das células e de reconhecimento das pistas. Este tempo, considerado elevado, se deve essencialmente a arquitetura do processador, que não foi projetado para realizar as operações matemáticas requeridas nestas etapas. Uma solução possível para esta inconveniência é a utilização de um processador dedicado especializado em processamento digital de sinais – um DSP (*Digital Signal Processor*) – para efetuar o processamento das imagens.

6 CONSIDERAÇÕES FINAIS E PROPOSTAS PARA TRABALHOS FUTUROS

Os avanços dos dispositivos de aquisição de imagens, aliados ao crescente aprimoramento do hardware computacional, estão impulsionando o desenvolvimento dos sistemas de visão de máquina. Estes sistemas apresentam-se atualmente como uma poderosa estratégia para solucionar problemas de naturezas diversas, em áreas como medicina, biologia e, em especial, na indústria.

Davies (2012) destaca, no entanto, que ainda há muito trabalho a se realizar na área. Mesmo alguns problemas fundamentais, como o reconhecimento de caracteres, precisam ter suas soluções aperfeiçoadas para que possam ser considerados resolvidos por completo.

Com o intuito de introduzir princípios, modelos e aplicações dos sistemas de visão, este trabalho apresentou o projeto de um robô capaz de solucionar jogos de Sudoku impressos em papel. Trata-se de um plotador cartesiano construído exclusivamente com peças de LEGO, controlado por um sistema de visão de máquina. O robô proposto captura imagens por meio de uma *webcam* e verifica se nelas existem *puzzles* Sudoku. Os jogos identificados são então interpretados e solucionados. Finalmente, o robô escreve as soluções no papel com uma caneta acoplada ao seu elemento terminal, completando os sudokus.

A estrutura mecânica do manipulador foi projetada utilizando-se a biblioteca LDraw e o MLCad, softwares de desenho e modelagem tridimensional específicos para criações com LEGO. O robô foi dimensionado de forma que seu elemento terminal pudesse alcançar a área equivalente a de uma folha A4.

O sistema de visão responsável por reconhecer os jogos de Sudoku foi desenvolvido utilizando-se a ferramenta MATLAB e validado por meio de um protótipo. Um conjunto de testes composto por 30 imagens de boa qualidade de sudokus foi processado por este protótipo e os resultados obtidos mostraram-se extremamente satisfatórios: todos os jogos foram reconhecidas com sucesso. Após, o sistema final foi portado para o *brick* EV3, processador utilizado para controle do robô, utilizando a linguagem Python.

Algoritmos computacionais para resolver jogos de Sudoku utilizados em solucionadores do estado da arte também foram discutidos. Uma vez que o algoritmo de *backtracking* mostrou-se suficientemente eficaz para a aplicação final, um estudo detalhado dos mesmos não foi desenvolvido.

O sistema de controle de movimento do robô foi elaborado. Para tanto, as relações de cinemática direta e inversa da estrutura mecânica do manipulador foram estabelecidas e analisadas.

Por fim, o robô foi montado. Os resultados obtidos em testes conduzidos mostraram-se satisfatórios, condizentes com os resultados esperados.

Deduz-se, assim, que o objetivo geral proposto de *"projetar um robô capaz de solucionar jogos de Sudoku impressos em papel"* foi alcançado. Os conceitos utilizados para desenvolver o sistema de visão do robô servem como base para diversas outras aplicações da área. De forma semelhante, os fundamentos de robótica e de controle empregados na construção do robô podem também ser utilizados no projeto de máquinas industriais reais.

6.1 TRABALHOS FUTUROS

A lista de trabalhos futuros é apresentada abaixo.

- Substituir as rodas utilizadas para movimentar os elos do robô por cremalheiras e engrenagens, aumentando a precisão dos movimentos do mesmo.
- Projetar um suporte mais robusto para fixação da *webcam* responsável por obter as imagens de entrada para o robô.
- Otimizar o sistema de visão responsável por reconhecer os jogos de Sudoku, reduzindo o tempo necessário para que o robô execute tal tarefa.
- Substituir o *brick* EV3 por um microprocessador especializado em processamento de sinais digitais (*Digital Signal Processor – DSP*), melhorando a performance do sistema de visão.
- Implementar os algoritmos *Dancing Links* e *Rule-based*, utilizados para resolver sudokus, e testá-los por meio de *benchmarks* em conjunto com o já implementado algoritmo de *Backtracking*, elegendo ao final o mais eficiente para emprego no sistema de controle do robô.

REFERÊNCIAS

BERGGREN, P.; NILSSON, D. *A study of Sudoku solving algorithms*. Dissertação (Mestrado) — KTH Royal Institute of Technology, 2012.

BOROVIKOV, E.; LANE, W. A survey of modern optical character recognition techniques (draft). *International Journal of Computer Applications*, v. 1, n. 301, p. 1–37, 2004.

BRIOT, S.; BONEV, I. Are parallel robots more accurate than serial robots? *Transactions of the Canadian Society for Mechanical Engineering*, v. 31, n. 4, p. 445–455, 2007.

BURGER, W.; BURGE, M. J. *Digital image processing : an algorithmic introduction using Java*. New York (N.Y.): Springer, 2008. (Texts in computer science). ISBN 978-1-84628-379-6. Disponível em: <http://opac.inria.fr/record=b1127349>.

CAMPION, G.; WANG, Q.; HAYWARD, V. The pantograph mk-ii: a haptic instrument. In: *Intelligent Robots and Systems, 2005. (IROS 2005). 2005 IEEE/RSJ International Conference on*. [S.l.: s.n.], 2005. p. 193–198.

CARROLL, J. *Machine vision companies reporting strong numbers for 2014*. Mar 2015. Disponível em: <http://www.vision-systems.com/articles/2015/03/machine-vision-companies-reporting-strong-numbers-for-2014.html>.

CHIAVENATO, I. *Introdução à teoria geral da administração:*. McGraw-Hill do Brasil, 1983. ISBN 9788520437926. Disponível em: <https://books.google.com.br/books?id=L7MbCgAAQBAJ>.

CORKE, P. *Robotics, Vision and Control: Fundamental Algorithms in MATLAB*. [S.l.]: Springer, 2011. (Springer Tracts in Advanced Robotics). ISBN 9783642201431.

DAUGMAN, J. G. *Computer Vision Lectures Notes*. 2010. Disponível em: <http://www.cl.cam.ac.uk/teaching/0809/CompVision/CompVisNotes.pdf>.

DAVIES, E. R. *Computer and Machine Vision, Fourth Edition: Theory, Algorithms, Practicalities*. 4. ed. [S.l.]: Academic Press, 2012. ISBN 9780123869081.

DU, S. et al. Automatic license plate recognition (alpr): A state-of-the-art review. *Circuits and Systems for Video Technology, IEEE Transactions on*, v. 23, n. 2, p. 311–325, Feb 2013. ISSN 1051-8215.

DUDA, R. O.; HART, P. E. Use of the hough transformation to detect lines and curves in pictures. *Commun. ACM*, ACM, New York, NY, USA, v. 15, n. 1, p. 11–15, jan. 1972. ISSN 0001-0782. Disponível em: <http://doi.acm.org/10.1145/361237.361242>.

ERCSEY-RAVASZ, M.; TOROCZKAI, Z. The chaos within sudoku. *CoRR*, abs/1208.0370, 2012.

ERTHAL, G. *Erthal, G. J. Notas de aula da disciplina reconhecimento de padrões. INPE, 2008.* [S.l.], 2008.

EV3DEV.ORG. *Documentation*. 2015. Disponível em: <http://www.ev3dev.org/docs/>.

FRANCO, N. M. B. *Cálculo numérico*. 1. ed. São Paulo, SP: Prentice-Hall Brasil, 2006. 520 p. ISBN 8576050870.

GAREY, M.; JOHNSON, D. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. [S.l.]: W. H. Freeman, 1979.

GHEDIRA, K.; DUBUISSON, B. *Constraint Satisfaction Problems*. [S.l.]: John Wiley & Sons, Inc., 2013. 224 p. ISBN 9781118574522.

GONZALEZ, R. C.; WOODS, R. E. *Digital Image Processing*. 3. ed. Upper Saddle River, NJ, USA: Prentice-Hall, 2008. ISBN 013168728X.

GONZALEZ, R. C.; WOODS, R. E.; EDDINS, S. L. *Digital Image Processing Using MATLAB*. Upper Saddle River, NJ, USA: Prentice-Hall, 2003. ISBN 0130085197.

GORDON, P.; LONGO, F. *Mensa Guide to Solving Sudoku*. 1. ed. [S.l.]: Sterling, 2006. 272 p. ISBN 1402740115.

GROUP, U. D. W. *USB Device Class Specifications*. 2015. Disponível em: <http://www.usb.org/developers/docs/devclass_docs/>.

IMPEDOVO, S.; OTTAVIANO, L.; OCCHINEGRO, S. Optical character recognition: a survey. *International Journal of Pattern Recognition and Artificial Intelligence*, v. 5, n. 1, p. 1–24, Jun 1991.

ISO. *Robots and robotic devices – Vocabulary*. Geneva, Switzerland, 2012.

KIERI, A. *Context Dependent Thresholding and Filter Selection for Optical Character Recognition*. 2012. 45 p. (UPTEC F, 12 036).

KNUTH, D. E. Dancing links. *Millennial Perspectives in Computer Science*, p. 159–187, nov 2000.

LDRAW. *What is LDraw?* Jun 2015. Disponível em: <http://www.ldraw.org/>.

LEGO. *History of LEGO Robotics*. Jun 2015. Disponível em: <http://www.lego.com/en-us/mindstorms/history>.

LOGITECH. *Logitech HD Webcam C270*. 2015. Disponível em: <http://www.logitech.com/pt-br/product/hd-webcam-c270>.

MARCATO, A. L. M. *Introdução à robótica*. 2012. Slides de aula, Universidade Federal de Juiz de Fora.

MCGUIRE, G.; TUGEMANN, B.; CIVARIO, G. *There is no 16-Clue Sudoku: Solving the Sudoku Minimum Number of Clues Problem*. 2012. Cite arxiv:1201.0749 Comment: 36 pages.

Merriam-Webster Online. *Merriam-Webster Online Dictionary*. 2009. Disponível em: <http://www.merriam-webster.com>.

MOLER, C. *The Origins of MATLAB*. 2004. Disponível em: <http://www.mathworks.com/company/newsletters/articles/the-origins-of-matlab.html>.

MOON, T.; GUNTHER, J. Multiple constraint satisfaction by belief propagation: An example using sudoku. In: *Adaptive and Learning Systems, 2006 IEEE Mountain Workshop on*. [S.l.: s.n.], 2006. p. 122–126.

MUDA, N. et al. Optical character recognition by using template matching (alphabet). In: *National Conference on Software Engineering & Computer Systems 2007 (NACES 2007)*. [S.l.: s.n.], 2007.

NAZ, S. et al. Challenges in baseline detection of cursive script languages. In: *Science and Information Conference (SAI), 2013*. [S.l.: s.n.], 2013. p. 551–556.

NIKLAS, K. *Unsupervised Post-Correction of OCR Errors*. Dissertação (Mestrado) — Leibniz-Universität Hannover, Hannover, Germany, 2010.

PACURIB, J.; SENO, G.; YUSIONG, J. Solving sudoku puzzles using improved artificial bee colony algorithm. In: *Innovative Computing, Information and Control (ICICIC), 2009 Fourth International Conference on*. [S.l.: s.n.], 2009. p. 885–888.

PRSALPHA. *Full Size PRSalpha CNC*. 2015. Disponível em: <http://www.shopbottools.com/mproducts/images/products-PRS.jpg>.

RAVANI, R.; NOORALISHAHI, P.; AMANI, A. A novel approach for persian/arabic intelligent word recognition. In: *Visual Information Processing (EUVIP), 2011 3rd European Workshop on*. [S.l.: s.n.], 2011. p. 292–297.

ROSARIO, J. M. *Robótica Industrial I: Modelagem, Utilização e Programação*. [S.l.]: Editora Baraúna, 2010.

SAUVOLA, J.; PIETIKäINEN, M. Adaptive document image binarization. *Pattern Recognition*, v. 33, p. 225–236, 2000.

SEZGIN, M.; SANKUR, B. Survey over image thresholding techniques and quantitative performance evaluation. *Journal of Electronic Imaging*, v. 13, n. 1, p. 146–168, 2004.

SHAFAIT, F.; KEYSERS, D.; BREUEL, T. Efficient implementation of local adaptive thresholding techniques using integral images. In: *Document Recognition and Retrieval XV*. [S.l.: s.n.], 2008. p. 68150.

SHIH, F. Y. *Image processing and mathematical morphology: fundamentals and applications*. Boca Raton, FL, USA: CRC Press, 2009. ISBN 9781420089431. Disponível em: <http://opac.inria.fr/record=b1128531>.

SOLOMON, C.; BRECKON, T. *Fundamentals of Digital Image Processing: A Practical Approach with Examples in Matlab*. [S.l.]: Wiley-Blackwell, 2010. ISBN-13: 978-0470844731. ISBN 0470844736.

SONKA, M.; HLAVAC, V.; BOYLE, R. *Image Processing, Analysis, and Machine Vision*. 4. ed. [S.l.]: CENGAGE-Engineering, 2014. ISBN 1133593607.

WILSON, R. *How to Solve Sudoku: A Step-by-Step Guide*. 1. ed. [S.l.]: Infinite Ideas, 2005. 124 p. ISBN 1904902626.

WILSON, R. The sudoku epidemic. *Focus*, v. 26, n. 1, p. 5–7, 2006.

APÊNDICE A – Materiais e Ferramentas Utilizados

Este apêndice apresenta os materiais, ferramentas e dispositivos utilizados no projeto e na construção do manipulador, incluindo componentes mecânicos e eletrônicos. Os softwares utilizados para realizar a modelagem e simulação do sistema de controle e da estrutura mecânica do robô também são apresentados.

A.1 LEGO MINDSTORMS EV3

LEGO Mindstorms é uma série de kits de robótica originada como fruto de uma parceria entre o Massachusetts Institute of Technology (MIT) e o grupo LEGO (LEGO, 2015). Os kits combinam sensores, atuadores e um controlador inteligente programável (*brick*) com componentes *LEGO Technic*, uma linha específica para a criação de modelos mais técnicos e complexos. Inicialmente voltados ao público infantil, com o intuito de introduzir crianças à área de robótica, os kits Mindstorms são atualmente utilizados como ferramenta para construção de protótipos por engenheiros ao redor de todo o mundo, devido ao poder de processamento de seus controladores e, principalmente, à facilidade para criar estruturas mecânicas complexas em pouco tempo. Último produto da série, o *LEGO Mindstorms EV3* foi lançado no final de 2013.

O *LEGO Mindstorms EV3* é a terceira geração do kit *LEGO Mindstorms*. O *brick* EV3 pode ser programado com o *LEGO MINDSTORMS EV3 Software*, um ambiente de programação *drag-and-drop* intuitivo baseado em blocos. Ele também é capaz de suportar outras linguagens de programação, tais como *Java* e *C*, desde que seu *firmware* seja substituído.

O *Mindstorms EV3 Brick* utiliza um processador ARM9 com 16 MB de memória flash e 64 MB de memória RAM. Conta com oito portas de conexão RJ-12, utilizadas para comunicação com sensores e atuadores, uma porta mini-USB para conexão com computador, um slot para cartão de memória microSD e uma porta USB de uso geral. Ele conta ainda com um display LCD monocromático, quatro botões de direção, um botão de voltar, um botão de confirmação e um pequeno auto-falante na lateral para reprodução de sons. A Figura 48 ilustra o *brick* EV3.

Figura 48 – Brick do kit LEGO Mindstorms EV3

Fonte: LEGO (2015)

O kit conta ainda com diversos sensores e atuadores que podem ser utilizados em conjunto com o controlador. Os sensores mais popularmente utilizados são os sensores de temperatura, luz, som, toque e ultra sônico. Entre os atuadores, podemos citar os motores de passo, servomotores e lâmpadas.

A.2 LEGO TECHNIC

LEGO Technic é uma linha de peças LEGO voltada para a criação de modelos mais complexos. Esta linha conta com peças como engrenagens, polias, motores elétricos, vigas, rodas e até mesmo atuadores pneumáticos.

As peças da linha *Technic* são muito utilizadas em conjunto com os kits *Mindstorms* para a construção de robôs, haja vista que as peças de linhas temáticas mais simples normalmente não são suficientes para montagem dos robôs desejados. A Figura 49 apresenta algumas engrenagens do *LEGO Technic*.

Figura 49 – Peças da linha LEGO Technic

Fonte: LEGO (2015)

A.3 LDRAW

O *LDraw*, abreviação para *LEGO Draw*, é uma suíte de bibliotecas e aplicativos *open-source* criada em 1995 especificamente para modelagem virtual em 3D de criações utilizando peças de LEGO (LDRAW, 2015). O LDraw permite ao usuário criar modelos e cenas virtuais, documentar construções, criar instruções de montagens, renderizar em 3D imagens dos modelos com qualidade fotográfica e criar animações.

O formato de arquivos proposto pelo LDraw tornou-se o padrão de fato para descrever modelos LEGO, sendo utilizado até os dias atuais (LDRAW, 2015). Contando com uma grande comunidade de contribuidores, a biblioteca possui muitas das peças oficiais LEGO, em especial de linhas técnicas como a *Technic* e os kits *Mindstorms*.

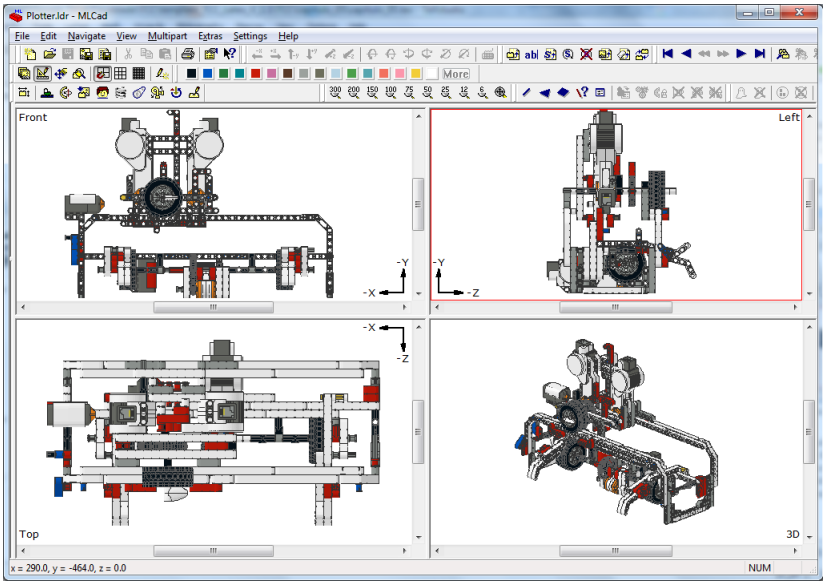
Normalmente, a biblioteca LDraw é utilizada em conjunto com um programa de computer-aided design (CAD) para desenho das criações, tal como o *MLCad*, e de um renderizador 3D, como o POV-Ray.

A.4 MLCAD

O *MLCad*, sigla para *Mike's LEGO CAD*, é um software de modelagem virtual CAD criado para desenhar criações com peças de

LEGO (LDRAW, 2015). Utilizando a biblioteca de peças fornecida pelo LDraw, ele permite criar modelos realistas de projetos que utilizam LEGO. A Figura 50 mostra o ambiente de desenho do programa MLCad.

Figura 50 – Ambiente do MLCad



Fonte: Interface do software no sistema operacional Windows 7

Além de disponibilizar ao usuário todas as peças da biblioteca LDraw, o MLCad possui também algumas ferramentas adicionais que permitem realizar modelagens bastante precisas. O programa também é capaz de executar verificações sobre o modelo, avaliando, por exemplo, a estabilidade estática de estruturas construídas.

A.5 POV-RAY

POV-Ray, abreviação de *Persistence of Vision Raytracer*, é um programa *open-source* de *ray tracing* que gera imagens de cenas a partir de descrições textuais. O *ray tracing* é uma técnica de renderização de imagens tridimensionais que simula o caminho que os raios de luz

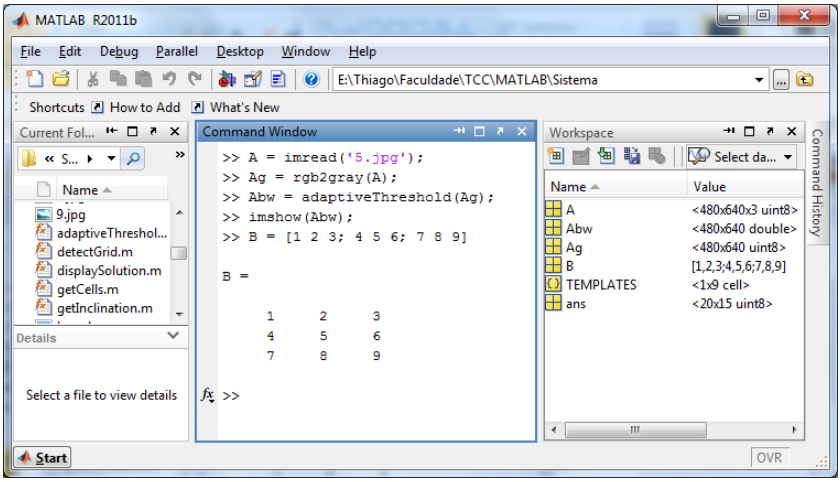
tomaria no mundo real, capaz de obter imagens com grande nível de realismo.

A biblioteca LDraw possui uma ferramenta chamada *L3P* que realiza a conversão de modelos em formato de arquivo LDraw para o formato das cenas utilizado pelo POV-Ray, possibilitando a obtenção de uma imagem próxima da realidade do modelo LEGO antes que ele seja de fato montado.

A.6 MATHWORKS MATLAB

O *MATLAB*, acrônimo para *Matrix Laboratory*, é uma linguagem de alto nível e um ambiente interativo de computação numérica. Ele integra ferramentas de computação, programação e visualização de dados em um ambiente de fácil utilização, onde problemas e soluções podem ser descritos rapidamente em notação matemática familiar. O MATLAB é uma ferramenta amplamente utilizada nas universidades – principalmente em cursos de engenharia – e na indústria, pois permite desenvolver pesquisas e análises numéricas em um tempo muito inferior ao que seria necessário utilizando outras linguagens de programação (MOLER, 2004). A Figura 51 apresenta o ambiente do MATLAB.

Figura 51 – Ambiente do MATLAB



Fonte: Interface do software no sistema operacional Windows 7

O MATLAB é utilizado principalmente para: realizar cálculos numéricos; desenvolver algoritmos; modelar, simular e prototipar sistemas; analisar, explorar e visualizar dados e; desenvolver aplicações (MOLER, 2004).

O ambiente conta ainda com famílias de ferramentas voltadas para aplicações específicas, chamadas de *toolboxes*. As *toolboxes* são coleções abrangentes de funções MATLAB que estendem o ambiente MATLAB para resolver classes particulares de problemas. Áreas que possuem *toolboxes* disponíveis incluem processamento de imagens e visão computacional, estatística e otimizações, processamento de sinais, sistemas de controle e muitas outras.

A *toolbox* de processamento de imagens possui diversos algoritmos prontos para utilização que são empregados com frequência na área, além de funções e aplicações para processamento, análise, desenvolvimento de outros algoritmos da área e visualização de resultados. São exemplos de algoritmos implementados a transformada de Hough, o *thresholding* pelo método de Otsu, transformações geométricas, diversas operações morfológicas e um amplo número de filtros.

A.7 DISTRIBUIÇÃO EV3DEV

O *ev3dev* é uma distribuição *Linux*, *open-source*, customizada especialmente para o brick programável do kit LEGO Mindstorms EV3. Ele é baseado no sistema operacional *Debian Jessie* e utiliza a versão 3.16 do kernel *Linux*. Diferentemente de demais firmwares para bricks *Mindstorms*, que são projetados para permitir a programação do brick em uma linguagem específica – tais como o *leJOS* e *NXC* –, a distribuição *ev3dev* foi desenvolvida com o intuito de permitir que desenvolvedores possam utilizar diversas linguagens de programação para comunicação com os periféricos LEGO.

O *ev3dev* é considerado um sistema operacional completo e foi inicialmente desenvolvido para a família ARM9, a qual pertence o processador do brick EV3. Ele foi posteriormente portado para outras famílias ARM e também é utilizado atualmente em outras plataformas, tais como o Raspberry PI em conjunto com o BrickPI.

Atualmente, o ev3dev possui APIs oficiais de comunicação com os periféricos LEGO para diversas linguagens de programação, tais como *C++, Python, Ruby, Node.js e Lua*. As APIs são desenvolvidas com base em uma especificação única, garantindo que todas as interfaces sejam praticamente idênticas para todas as linguagens suportadas.

O usuário pode ainda utilizar outras linguagens de programação, criando suas próprias bibliotecas. Para tanto, basta que a linguagem possua *port* para a arquitetura alvo ARM e seja capaz de realizar operações de leitura e escrita em arquivos – a comunicação com os componentes LEGO é realizada por meio de operações de *I/O* em *device files*, tal como acontece nos sistemas operacionais **NIX* de computadores pessoais para comunicação com impressoras e portas seriais, por exemplo. A instalação de linguagens e ferramentas de suporte para execução de programas desenvolvidos nas mesmas pode ser realizada por meio do *Advanced Packaging Tool (apt)*, que é suportado pela distribuição.

O *ev3dev* possui drivers para a grande maioria dos periféricos LEGO dos kits Mindstorms e também para componentes de outros fabricantes. Os drivers implementam todas as funções básicas do firmware original, tais como leitura de sensores (temperatura, luz, som, toque e ultrasônico) e controle de atuadores (motores, LEDs, display LCD e auto-falante). Uma lista completa dos sensores e atuadores suportados pode ser encontrada em (EV3DEV.ORG, 2015).

O *ev3dev* pode ser utilizado a partir de um cartão microSD, mantendo o *firmware* original do EV3 intacto. Para tanto, basta instalá-lo em um cartão e inserir este no slot microSD do *brick*. O *bootloader* irá identificar e inicializar o sistema *ev3dev* quando o *brick* for ligado. Para voltar a utilizar o *firmware* original, basta desligar o controlador e remover o cartão microSD do mesmo – o próximo *boot* será então realizado com o firmware do original.

A.8 WEBCAM LOGITECH C270

A *Logitech C270*, exibida na Figura 52, é uma *webcam USB* que captura fotos com resolução de até **3.0** megapixels e é compatível com o *USB Video Device Class (UVC)*.

O *UVC* é uma especificação universal que descreve um protocolo específico para transferência de vídeo por meio da interface USB (GROUP, 2015). As classes de dispositivo USB, como o UVC, provém independência do *host* para suportar dispositivos de diferentes fabricantes. A grande vantagem de utilizar uma webcam que compatível com a especificação *UVC* é a garantia de seu funcionamento em grande parte dos sistemas operacionais populares, uma vez que praticamente todos eles a implementam. O *Linux*, por exemplo, inclui o driver para dispositivos *UVC* na distribuição de seu kernel desde a versão *2.5.26*,

tornando-a compatível com o *ev3dev* e outras distribuições que utilizam um kernel *Linux* superior a esta versão.

Figura 52 – Webcam Logitech C270

Fonte: Logitech (2015)