GLÊNIO SILVA PIMENTEL

DESENVOLVIMENTO DE UMA APLICAÇÃO *ANDROID* PARA MAPEAMENTO DE UMA MESA COM OBSTÁCULOS E ROBÔS *E-PUCK* UTILIZANDO VISÃO COMPUTACIONAL

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE ENGENHARIA MECÂNICA
2017

GLÊNIO SILVA PIMENTEL

DESENVOLVIMENTO DE UMA APLICAÇÃO *ANDROID* PARA MAPEAMENTO DE UMA MESA COM OBSTÁCULOS E ROBÔS *E-PUCK* UTILIZANDO VISÃO COMPUTACIONAL

Trabalho de Conclusão de Curso (TCC) apresentado à Faculdade de Engenharia Mecânica da Universidade Federal de Uberlândia, como requisito à obtenção do título de Bacharel em Engenharia Mecatrônica.

Orientador: Prof. Dr. Mauricio Cunha Escarpinati

Uberlândia – MG

Julho de 2017

DEDICATÓRIA

À minha família e à minha namorada,
pelo carinho e apoio durante
as etapas de minha vida.

O problema não é o problema.
O problema é sua atitude em relação ao problema.
Você entende?

Jack Sparrow
(Piratas do Caribe: A maldição do Pérola Negra)

AGRADECIMENTOS

Em especial, agradeço primeiramente a Deus, pelo dom da vida que me é dado todos os dias e a todas as pessoas em minha volta, permitindo que possamos sorrir com saúde e determinação, celebrando juntos as belezas da vida.

Agradeço também a todos aqueles que diretamente ou indiretamente me apoiam ou me apoiaram nas etapas de minha vida, como na minha educação e conclusão deste trabalho. Assim, agradeço:

Aos meus pais, Eni Maria da Silva Pimentel e Gilmar dos Santos Pimentel, por nunca deixarem faltar nada em minha vida, por me darem amor, carinho, educação, orientação, apoio nos momentos difíceis, e não menos importante, por me ensinarem a valorizar a vida e ao próximo, e a lutar por tudo aquilo em que acreditar.

Ao meu irmão, Sávio Silva Pimentel, pelos momentos de diversão e crescimento juntos, e aos meus avós, pela sua sabedoria e carinho. Em particular agradeço à minha avó Maria Aparecida da Silva, que mesmo não estando mais presente, ainda me inspira a sonhar. Sinto saudade sua vó, obrigado por tudo.

Aos meus primos(as) e tios(as) por me propiciarem momentos de diversão, almoços em família, e por todo companheirismo, mesmo nas horas difíceis.

À minha namorada, Camille Nunes Leite, por todo amor, carinho e companheirismo, e por me ensinar a apreciar cada detalhe da vida, pois tudo tem seu valor e é belo a sua maneira. Você me inspira ainda mais a lutar pelos meus sonhos, incluindo você.

A todos meus amigos e à família de minha namorada, por compartilharem tantos momentos da vida de vocês comigo, tem sido um prazer.

Ao meu orientador, Prof. Dr. Mauricio Cunha Escarpinati, pelo apoio, pelas dicas dadas, e por acreditar na conclusão deste trabalho.

À Universidade Federal de Uberlândia e à Faculdade de Engenharia Mecânica pela oportunidade de realizar esse curso.

Por fim, a todos aqueles que contribuíram com minha formação, desde meus professores do jardim de infância, a todos os contribuintes deste país. Espero poder retribuir com meus conhecimentos para a construção de um país melhor e mais justo.

PIMENTEL, G. S. Desenvolvimento de uma aplicação *Android* para mapeamento de uma mesa com obstáculos e robôs *E-Puck* utilizando visão computacional. 2017. 92 f. Trabalho de Conclusão de Curso, Universidade Federal de Uberlândia, Uberlândia.

RESUMO

A necessidade crescente de tecnologias de alta precisão, associadas a uma elevada repetitividade e velocidade na execução de tarefas diversas, tem ocasionado no crescimento acelerado da robótica ao longo dos anos, e em particular, da robótica Móvel. Dentre os inúmeros desafios dessa área está a necessidade do cálculo do menor trajeto a ser percorrido de um ponto a outro, e que este esteja livre de obstáculos, de modo a se alcançar um objetivo definido. Com base nisso, o planejamento de trajetória é fundamental em robótica. Diversas teorias têm sido aplicadas no estudo do planejamento de trajetória, como grafos de visibilidade e autômatos celulares, esta última baseada em decomposição celular. Neste contexto, diversas são as abordagens empregadas no tratamento de problemas envolvendo planejamento de trajetória. Uma das abordagens possíveis é sua integração com técnicas de visão computacional. Deste modo, este trabalho propõe o desenvolvimento de uma aplicação *Android* baseada em técnicas de visão computacional para mapeamento de uma mesa, ou bancada, de testes com obstáculos, robôs *E-Puck* e um objetivo. Assim, um ambiente real de testes foi proposto, no qual estudos de planejamento de trajetória possam ser realizados com robôs móveis em futuros trabalhos. Um ambiente de testes deste tipo é bastante vantajoso, em uma área de estudos que geralmente envolve equipamentos de alto custo ou disponíveis apenas na forma de ambientes de simulação, que apesar de fornecerem bons resultados, ainda necessitam de verificação em um ambiente real para validação dos resultados. Ao final deste trabalho, uma aplicação *Android* flexível e configurável é apresentada, desenvolvida com a utilização de técnicas de filtragem de cor, filtragem de ruído, operações morfológicas, identificação de objetos e simplificação por polígonos. Além disso, a biblioteca de desenvolvimento *OpenCV* foi integrada à aplicação, por conter uma implementação madura e otimizada das técnicas mencionadas. Por se tratar de um primeiro estudo, melhorias foram também propostas, bem como sugestões de trabalhos futuros.

Palavras Chave: Mapeamento. Visão Computacional. E-Puck. Android. Robótica.

PIMENTEL, G. S. Development of an Android application for mapping a table with obstacles and E-Puck robots using computer vision. 2017. 92 f. Undergraduate Thesis, Universidade Federal de Uberlândia, Uberlândia.

ABSTRACT

The increasing need for high precision technologies, coupled with high repeatability and speed in performing various tasks, has led to the accelerated growth of robotics over the years, and in particular mobile robotics. Among the numerous challenges in this area it is the need to calculate the shortest path to be traveled from one point to another, free of obstacles, in order to achieve a defined goal. Based on this, the path planning is fundamental in robotics. Several theories have been applied in the study of path planning, such as visibility graphs and cellular automata, the latter based on cellular decomposition. In this context, several approaches are used in the treatment of problems involving path planning. One of the possible approaches is its integration with computer vision techniques. In this way, this work proposes the development of an Android application based on computer vision techniques for mapping a testing table, or testing bench, with obstacles, E-Puck robots and a goal. Thus, a real testing environment has been proposed, in which path planning studies can be performed with mobile robots in future works. Such a testing environment is quite advantageous in a field of studies that usually involves high cost equipment or only available in the form of simulation environments which, while providing good results, still requires verification in a real environment for validation of results. At the end of this work, a flexible and configurable Android application is presented, developed with the use of color filtering techniques, noise filtering, morphological operations, identification of objects and simplification by polygons. In addition, the OpenCV development library has been integrated into the application, as it contains a mature and optimized implementation of mentioned techniques. As it is a first study, improvements are also proposed, as well as suggestions for future works.

Keywords: Mapping. Computer Vision. E-Puck. Android. Robotics.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SÍMBOLOS E ABREVIAÇÕES

θ – ângulo das equações paramétricas da Transformada Circular de *Hough*

π – número *pi*

σ – desvio padrão

3G – conectividade móvel sem fio de terceira geração

4G – conectividade móvel sem fio de quarta geração

(a, b) – coordenadas do centro do círculo nas equações paramétricas da Transformada Circular de *Hough*

A/D – Analógico/Digital

API – Application Programming Interface

ARM – Advanced RISC Machine

bit – binary digit

CCD – Charge-Coupled Device

chip – contração para circuito integrado

CMOS – Complementary metal–oxide–semiconductor

e – número de *Euler*

FACOM – Faculdade de Computação

$G(i, j)$ – filtro (ou kernel) gaussiano aplicado como uma janela de coordenadas (i, j)

g – grama

GB – Gigabyte

GHz – Gigahertz

h – metade do comprimento ou lado da janela do kernel de convolução

HSB – Hue, Saturation and Brightness (espaço de cor)

HSI – Hue, Saturation and Intensity (espaço de cor)

HSV – Hue, Saturation and Value (espaço de cor)

(i, j) – par de coordenadas da matriz do kernel de convolução

I – imagem de entrada na operação de convolução

IA – Inteligência Artificial

IDE – Integrated Development Environment

IBM – International Business Machines

JNI – Java Native Interface

K *– kernel de convolução*

LED – Light Emitting Diode

Li-ion – lithium-ion

$O[u, v]$ – imagem de saída pós operação de convolução

m – metros

MAC – Media Access Control

mAh – miliampère-hora

MDF – Medium Density Fiberboard

MIT – Massachusetts Institute of Technology

mm – milímetro

MP – Megapixel

NDK – Native Development Kit

nm – nanômetro

OpenCV – Open Source Computer Vision Library

PC – Personal Computer

PDI – Processamento Digital de Imagens

pixel – picture element

pol – polegada

PVC – Policloreto de vinila

qtd – quantidade

R – raio do círculo nas equações paramétricas da Transformada Circular de *Hough*

RGB – Red, Green and Blue (espaço de cor)

RGBA – Espaço de cor *Red, Green e Blue*, mais *Alpha* (canal transparente)

Câmera *RGB-D* – Câmera com canais *Red, Green* e *Blue*, mais profundidade (do inglês, *Depth*)

RFCOMM – Radio frequency communication

SDK – Standard Development Kit

SO – Sistema Operacional

SPP – Serial Port Profile

(u, v) – par de coordenadas horizontal e vertical de uma imagem

UFU – Universidade Federal de Uberlândia

UUID – Universally Unique Identifier

USB – Universal Serial Bus

VVA – Vermelho, Verde e Azul

W – janela do kernel de convolução

WiFi – protocolo de comunicação sem fio

WLAN – Wireless Local Area Network

SUMÁRIO

CAPÍTULO I

INTRODUÇÃO

Em 2015, o mercado da robótica foi responsável por movimentar 71 bilhões de dólares, e de acordo com a empresa internacional de consultoria em inteligência de mercado, *International Data Corporation*, é um mercado que deverá atingir em 2019, a marca de 135,4 bilhões de dólares (VANIAN, 2016).

O mercado da robótica cresce a cada ano e seu sucesso se deve às diversas áreas em que é aplicada. Robôs são responsáveis por inúmeras tarefas na indústria manufatureira, empregados em atividades de soldagem, pintura, transporte, manipulação de componentes de eletrônica e até mesmo de fármacos, realizando todas estas tarefas durante horas e com precisão, repetitividade e velocidade além das capacidades humanas. Contudo, a maioria destes robôs realizam suas tarefas em uma posição fixa, ou seja, geralmente se trata de um robô formado por um braço robótico e uma base, onde apenas o braço tem liberdade para se movimentar em seu volume de trabalho, como um robô típico da Indústria automobilística, apresentado na Fig. 1.1. Assim, a robótica tem procurado compensar uma das suas grandes desvantagens, a falta de mobilidade no ambiente em volta do robô (SIEGWART; NOURBAKHSH, 2004).

Essa nova abordagem da robótica é chamada de robótica móvel. Um robô móvel possui a capacidade de percorrer toda uma planta fabril, realizando suas tarefas com base em uma programação mais flexiva e adaptativa, conforme as necessidades (SIEGWART; NOURBAKHSH, 2004). Robôs do tipo móvel estão cada vez mais presentes, não apenas no meio fabril, mas também em residências, hospitais, galpões de armazenamento e diversos outros ambientes. Como exemplo, a Figura 1.2 ilustra a aplicação dos robôs móveis *Kiva* da empresa norte-americana *Amazon* em um dos seus armazéns. Esses robôs, cerca de 45 mil

no início de 2017, são responsáveis pela movimentação de milhares de prateleiras de mercadorias dentro dos vários armazéns da empresa (SHEAD, 2017).

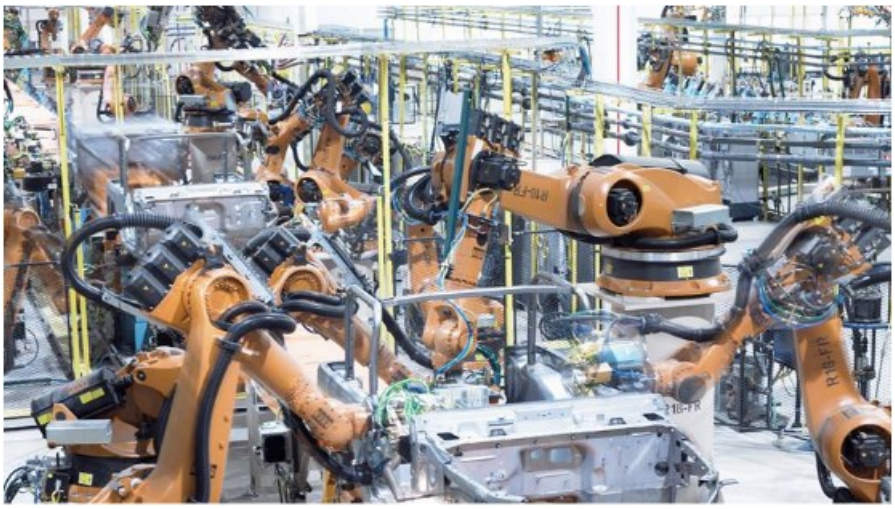


Figura 1.1. Típico robô da empresa KUKA usado na Indústria automobilística (KUKA, 2017)

Figura 1.2. Robôs móveis responsáveis pela movimentação de inúmeras prateleiras de produtos em um dos armazéns da empresa *Amazon* (BERGER, 2014)

Contudo, a robótica móvel constitui um grande desafio, principalmente em situações onde o ambiente não é estruturado, fornecendo diversos obstáculos ao robô, por vezes não previstos. Assim, duas linhas importantes de pesquisa para robótica móvel são o

planejamento de trajetória e o desvio de obstáculos. O planejamento de trajetória tem sido extensivamente estudado por causa de sua grande importância. Mesmo os típicos robôs industriais, como os da Fig. 1.1, necessitam de um planejamento de trajetória para movimentação do seu braço robótico. Diversas abordagens têm sido desenvolvidas para solucionar o planejamento de trajetória, dentre elas: o mapa de estradas (*road map*), a decomposição celular (*cell decomposition*) e o campo potencial (*Potential field*). Em especial, a decomposição celular tem como objetivo a realização do planejamento de uma trajetória com base em células, que geralmente podem estar em dois estados: livres e ocupadas por objetos. A ideia é dividir o caminho a ser seguido pelo robô nestas diversas células e atribuir estes estados às células (SIEGWART; NOURBAKHSH, 2004). Dentro do contexto da decomposição celular, inúmeras técnicas podem ser aplicadas, como as baseadas na teoria de autômatos celulares, que tem sido aplicada como um meio efetivo na busca por uma trajetória ótima a ser percorrida em um ambiente com obstáculos, a partir de pontos de partida e destino definidos (BEHRING et al., 2000).

Não obstante, diversas outras metodologias e técnicas de apoio podem ser empregadas no estudo do planejamento de trajetória baseado em decomposição celular. Uma das abordagens possíveis é a integração de técnicas de visão computacional, fornecendo um mecanismo de auxílio automático à análise e controle da execução dos algoritmos de planejamento de trajetória através de inspeção e *feedback* visual. Dentro da proposta baseada em visão computacional, alguns objetivos foram definidos para este trabalho.

1.1 Objetivos

Os objetivos deste trabalho podem ser divididos em:

Objetivo Geral:

Desenvolver uma aplicação *Android* para utilização com uma mesa ou bancada de testes de robôs, que forneça um mapeamento dos objetos na bancada através de técnicas de visão computacional. Na bancada são colocados robôs do tipo *E-Puck* (EPFL, 2017), um objetivo a ser atingido, e alguns obstáculos. Assim, com o mapeamento destes objetos na bancada, técnicas de planejamento de trajetória em robótica móvel, especialmente baseadas

em decomposição celular, poderão ser estudadas e aplicadas em um ambiente real de testes. Cabe ressaltar que os termos mesa e bancada são usados de forma equivalente neste trabalho.

Objetivos Específicos:

- Estudo das técnicas de visão computacional;
- Estudo da biblioteca *OpenCV* para implementação das técnicas de visão computacional;
- Estudo do sistema operacional *Android* e o desenvolvimento de uma aplicação neste ambiente;
- Integração dos componentes de *software*, *OpenCV* e *Android*;
- Estudo do protocolo de comunicação *Bluetooth* para possibilidade de transmissão dos mapas gerados pela aplicação para dispositivos remotos clientes;
- Teste da aplicação desenvolvida, geração do mapa e coleta de resultados;
- Avaliação dos resultados colhidos e verificação da acurácia e funcionalidade geral da aplicação no processo de mapeamento;
- Avaliação das possíveis modificações futuras a serem realizadas na aplicação desenvolvida.

1.2 Justificativa

Estudos com robôs geralmente são realizados em ambientes de simulação, onde diversos algoritmos e configurações possam ser testados sem o gasto dispendioso de um ambiente de desenvolvimento real, em especial devido à aquisição dos robôs e dos componentes de *hardware* necessários, ou da falta de um ambiente adequado disponível. Como exemplos, pode se citar os ambientes de simulação *Webots* (CYBERBOTICS LTD., 2017) e *V-REP* (COPPELIA ROBOTICS, 2017) utilizados por Ferreira; Vargas; Oliveira (2014) e Oliveira; Vargas; Ferreira (2015), respectivamente, para estudos de planejamento de trajetória utilizando autômatos celulares. Contudo, ambientes de simulação não são perfeitos, dependendo, entre outras coisas, do nível de fidelidade na modelagem dos componentes inseridos nesses ambientes. Assim, o ambiente ideal de estudo constituiria o conjunto

formado por simulador e uma bancada ou ambiente de testes real, para validação dos resultados obtidos em simulação. Com isto em mente, uma base de experimentação real utilizando captura de imagens, e com a capacidade de fornecer dados de mapeamento dos objetos presentes na bancada de forma automática e confiável através de técnicas de visão computacional, constitui um projeto justificável a ser desenvolvido.

Não obstante, foi proposto a utilização de um *smartphone* embarcado com o sistema operacional *Android*, como *hardware* e plataforma de desenvolvimento responsável pela captura e processamento das imagens. O sistema operacional *Android* é o sistema operacional para dispositivos móveis mais utilizado no mundo, e sua escolha para este trabalho, além de sua popularidade, se deve às suas características robustas e a um ambiente rico de desenvolvimento. Já os *smartphones* são dispositivos móveis que podem ser considerados computadores de bolso, fornecendo mais poder de processamento do que toda a plataforma computacional disponível pela agência aeroespacial do governo norte-americano, *NASA*, quando esta enviou o primeiro homem à lua em 1969 (KHOSO; KHAN, 2016).

Ainda segundo Khoso e Khan (2016), em 2016 haviam cerca de 2 bilhões de *smartphones* no mundo e um crescimento esperado de 4 bilhões para 2020, constituindo um dispositivo bastante presente em nossas vidas. Alguns *smartphones* atuais estão disponíveis por preços similares aos dos celulares normais, constituindo uma plataforma de baixo custo, fornecendo funcionalidades similares à um computador pessoal (*PC*), e permitindo ainda que seus usuários realizem diversas tarefas, como: acesso à internet, captura de fotos e vídeos, entretenimento através de jogos, produtividade através de aplicações de calculadora e editores de texto, serviços de telefonia móvel e de mensagem, e várias outras. Além disso, *smartphones* estão disponíveis com uma gama de sistemas operacionais semelhantes aos dos computadores pessoais, destacando-se entre estes sistemas, o sistema operacional *Android* da empresa *Google*, o *iOS* da empresa *Apple* e o *Windows Mobile* da empresa *Microsoft*. Estes aparelhos geralmente possuem conectividade *USB (Universal Serial Bus)*, *Bluetooth*, *3G* e *4G*, *WiFi*, e podem vir embarcados com diversos tipos de sensores, desde acelerômetros à bússola (BARROS, 2012).

Desta forma, e principalmente devido à sua mobilidade, o *smartphone* embarcado com o sistema operacional *Android* reforça a justifica de uma plataforma de captura de imagens para utilização com o ambiente de testes mencionado.

1.3 Organização do trabalho

Este trabalho está organizado na forma de capítulos, contendo sete capítulos incluindo esta Introdução. Estes capítulos são:

- Capítulo II: discute as diferenças entre as áreas de processamento digital de imagens e visão computacional, abordando as técnicas compreendidas por estas áreas. O capítulo traz ainda uma breve descrição do ambiente de desenvolvimento *Android*;
- Capítulo III: apresenta trabalhos correlatos a este, identificando metodologias e técnicas similares utilizadas por outros autores;
- Capítulo IV: apresenta a aplicação *Android* desenvolvida e sua integração com a biblioteca *OpenCV*, explicando o funcionamento e as particularidades da aplicação;
- Capítulo V: discute a metodologia utilizada na realização deste trabalho, informando os materiais utilizados e as abordagens empregadas;
- Capítulo VI: apresenta e discute todos os resultados obtidos nos testes realizados com a aplicação *Android* e a bancada de testes com os objetos em campo;
- Capítulo VII: conclui este trabalho e sugere trabalhos futuros a serem realizados.

CAPÍTULO II

REVISÃO BIBLIOGRÁFICA

Para o desenvolvimento do trabalho proposto, diversas técnicas e ferramentas são necessárias. Dessa forma, este capítulo descreve as técnicas de processamento digital de imagens (PDI), base da funcionalidade da aplicação, e o ambiente de desenvolvimento *Android*, fornecendo, em conjunto, uma *interface* gráfica e operacional para acesso à câmera do dispositivo *Android*, e permitindo o processamento das informações recebidas através da câmera.

2.1 Processamento digital de imagens e visão computacional

Uma imagem é dita digital se pode ser descrita por um conjunto finito de *pixels*, uma contração para *picture elements*, ou seja, elementos de imagem. Cada *pixel* representa uma posição discreta *u* e *v* (ou par de coordenadas (*u*, *v*)) no espaço bidimensional da imagem, sendo *u* e *v* perpendiculares entre si e representando, geralmente, as direções horizontal e vertical nesse espaço, respectivamente. Cada par de coordenadas (*u*, *v*) em uma imagem em escala de cinza possui um valor, chamado de intensidade ou nível de cinza, informando a quantidade de luz discretizada presente naquele ponto da imagem. No caso de uma imagem colorida, cada par de coordenadas (*u, v*) pode estar associado a mais de um valor, conforme espaço de cor utilizado na representação da imagem, e explicado mais adiante. Assim, esse conjunto finito de par de coordenadas discretas (*u, v*) e seus respectivos valores de intensidade representam uma imagem, que por sua vez pode ser colocada na forma de uma

matriz com as colunas representando a coordenada *v* e as linhas representando a coordenada *u*, e cada valor da matriz representando a intensidade na sua respectiva linha e coluna ou par de coordenadas (*u*, *v*) (WOODS; GONZALEZ, 2008). Dessa forma, uma imagem é tratada por um sistema computacional na forma de uma matriz de números, como apresentado na Fig. 2.1. Todas as técnicas de PDI apresentadas neste trabalho atuam sobre esse pares de coordenadas e valores de intensidade.

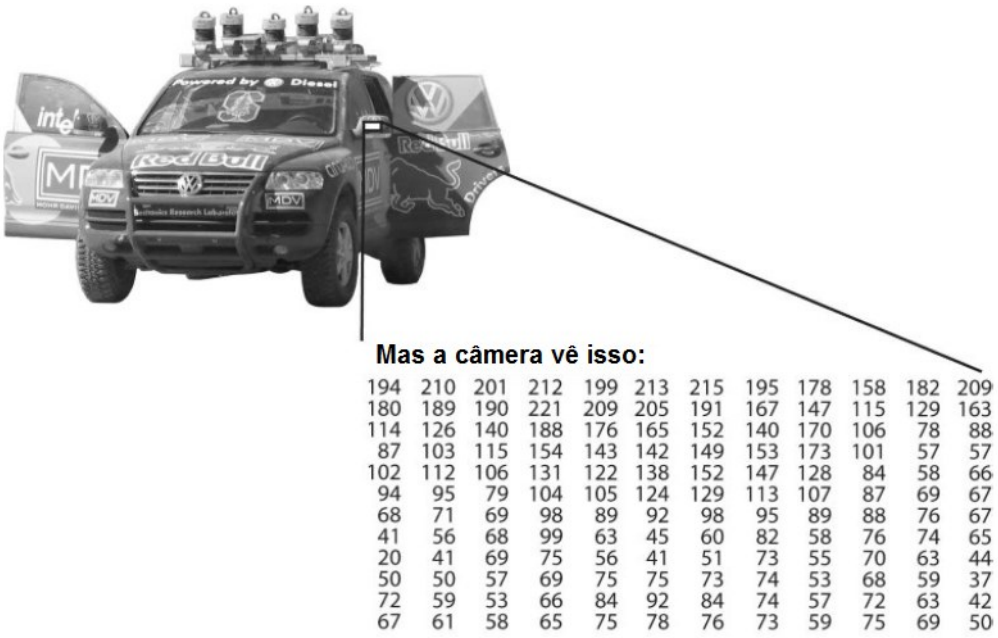


Figura 2.1. Para o computador, o retrovisor lateral de um carro é apenas uma matriz de números (Traduzido de KAEHLER; BRADSKI, 2017)

Segundo Woods e Gonzalez (2008), existem diversas áreas relacionadas ao PDI, como a análise de imagens e a visão computacional. Porém, não existe um consenso entre os diversos autores destas áreas sobre onde uma termina e a outra começa. Uma forma de tentar diferenciá-las seria dividir os processos realizados nessas áreas em três tipos. O primeiro processo, baixo nível, considera que a entrada do processo é uma imagem, onde esta imagem possa ser trabalhada para, por exemplo, realçar o contraste ou reduzir o ruído da mesma, tendo como saída, outra imagem processada. Esse primeiro processo seria mais próximo do conceito puro de processamento de imagem. O segundo processo, nível intermediário, considera como entrada uma imagem que poderá ser processada de forma a se extrair objetos, contornos ou outras características similares, fornecendo como saída uma entidade. O terceiro processo, alto nível, busca procurar características com base nos objetos e

contornos reconhecidos e relacioná-las com um conhecimento prévio de forma que essas características possam realmente serem interpretadas, por exemplo, reconhecer e identificar um trecho de rodovia como sendo uma estrada, e repassar essa informação a um veículo com navegação autônoma. Portanto, o processo em alto nível estaria mais próximo da área de visão computacional, abrangendo, geralmente, características e recursos da área de inteligência artificial (IA).

Por fim, Woods e Gonzalez (2008) consideram uma espécie de junção entre os processos de baixo nível e nível intermediário como a área de processamento digital de imagens. Assim, todo processo com uma entrada e uma saída do tipo imagem onde se possa filtrar a informação e extrair-se características na forma de objetos individuais, trata-se de um processamento digital de imagens. Neste trabalho, o processamento digital de imagens é considerado como parte integrante da área de visão computacional, sendo englobado por ela. Deste modo, os subtópicos a seguir apresentam as técnicas de PDI utilizadas neste trabalho, e o capítulo VI apresenta uma breve discussão sobre a utilização do conceito de visão computacional como tema deste trabalho.

2.1.1 Filtragem de cor

Humanos e outros animais frequentemente analisam cenas ou imagens por meio das diferentes cores presentes nelas. Objetos de diferentes formas e tamanhos podem ser distinguidos entre si e reconhecidos por meio dessas cores. Assim, os objetos presentes na mesa possuem diferentes cores com o intuito de facilitar a sua identificação, de forma que o primeiro passo do mapeamento é separar o objetivo e as bordas do campo na mesa através da filtragem da cor vermelha presente nestes objetos, e posteriormente, separar os obstáculos através da filtragem da cor preta.

Humanos são tricromatas, ou seja, possuem três tipos de receptores (ou cones) no globo ocular, mais especificamente na região da retina, que respondem por três diferentes partes do espectro. Essas três partes diferentes do espectro têm seu pico de resposta no olho humano entre o que se denomina, as cores: vermelho, verde e azul; constituindo as três cores primárias (CORKE, 2011). Na Figura 2.2 é possível visualizar o esquema de representação de cores de acordo com a frequência do espectro eletromagnético. Observa-se que a luz visível, da cor roxa à cor laranja, compreende a faixa do espectro com comprimento de onda de aproximadamente 400 nm à 700 nm.

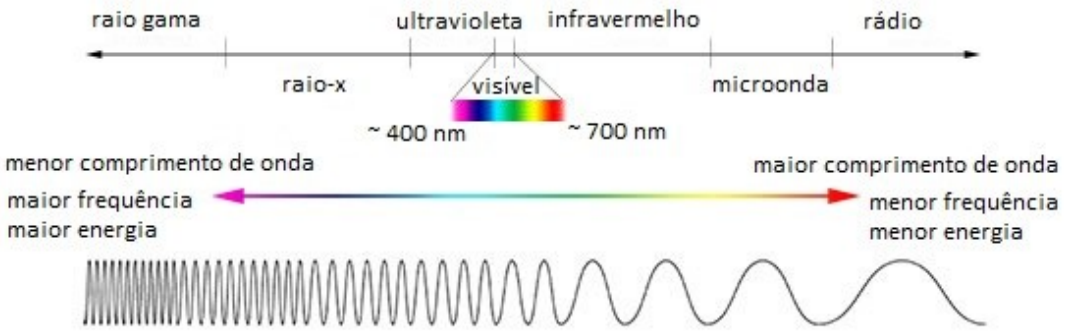


Figura 2.2. Comparação entre os comprimentos de onda, frequência e energia do espectro eletromagnético (Adaptado e Traduzido de UNIVERSE, 2013)

Um dispositivo computacional é capaz de representar e utilizar uma quantidade variável das cores primárias processadas pelo olho humano, constituindo um sistema do tipo VVA (Vermelho, Verde e Azul) ou como é comumente conhecido, o sistema *RGB* (inglês para: *Red*, *Green* e *Blue*). Porém, apesar da sua popularidade, o espaço cromático formado no sistema *RGB* não leva em consideração o brilho ou luminância de uma cor, sendo dificilmente processado e distinguido para diferentes condições de iluminação, fornecendo por vezes resultados diferentes para uma mesma cor. Além disso, humanos possuem uma maior facilidade em distinguir o espaço cromático, ou cromaticidade, em termos de matiz e saturação. Esses dois termos servem para diferenciar uma cor da outra levando em consideração a cor dominante presente, também chamada de matiz ou tonalidade, e o quão puro uma cor é, ou seja, uma cor que possua pouca quantidade de cor (ou luz) branca misturada à ela (já que a luz branca é uma mistura de todo o espectro visível), característica essa chamada de saturação (CORKE, 2011). Na Figura 2.3 são ilustradas as diferenças em se utilizar matiz e saturação. A Figura 2.3a apresenta uma cena original com flores, posteriormente submetida a um processo de filtragem da matiz vermelha, destacando apenas as flores vermelhas na Figura 2.3b. Já a Figura 2.3c representada a imagem original apenas com o nível de saturação, onde as flores brancas têm baixa saturação (elas parecem escuras).

Deve se mencionar ainda a existência de diversos espaços de cor ou sistemas de cor para representar uma cor nos termos de matiz e saturação. Neste trabalho, o espaço de cor *HSV* (*Hue*, *Saturation and Value*) foi escolhido, especialmente por sua popularidade e adoção em *softwares* de processamento de imagem e *design* gráfico, e por sua facilidade de utilização.

Figura 2.3. Cena com flores. a) Cor original; b) Imagem com matiz vermelha; c) Saturação da Imagem (CORKE, 2011)

Como mencionado anteriormente, o espaço de cor *HSV* é definido por três componentes: Matiz (ou tonalidade), Saturação e Valor (do inglês: *Hue, Saturation and Value*), sendo este último componente substituído às vezes por Intensidade (do inglês, *Intensity*) ou Brilho (do inglês, *Brightness*), se tornando sistemas *HSI* e *HSB*, respectivamente. A Figura 2.4 retrata o espaço de cor *HSV* como um cone. Na figura, a matiz é representada em cada seção circular do cone como um ângulo de zero a 360º. O ângulo de zero grau, ou matiz zero, equivale à uma tonalidade de vermelho. O ângulo de 240 graus, corresponde à uma tonalidade de azul e assim por diante. Com relação à distância do centro da seção circular do cone, obtém-se a saturação, assim, nota-se que quanto mais distante do centro da seção, maior a saturação, que vai de zero a cem. Dessa forma, uma saturação zero equivale à cor branca, e conforme a saturação cresce, tons de cinza para uma dada matiz correspondente são observados, até se atingir a saturação total, sem mistura de branco. Já a vertical do cone, ou seja, seu comprimento, informa o brilho, valor ou intensidade de um dado par formado por matiz e saturação, sendo que na maior seção do cone, o brilho é máximo (igual a cem) e na menor seção, todas as cores são escuras, ou preto (igual a zero) (CARDANI, 2007).

A Figura 2.5 apresenta de forma mais clara as seções circulares do cone representante do espaço *HSV*, destacando ainda as diferentes matizes e saturações conforme se circula pela seção do cone e conforme se caminha pela distância entre o centro do cone e sua borda. Em especial, demonstra as tonalidades de cinza dadas pela saturação. Observa-se também o efeito do componente de intensidade, ou valor, desde a menor seção do cone até a maior seção do mesmo.

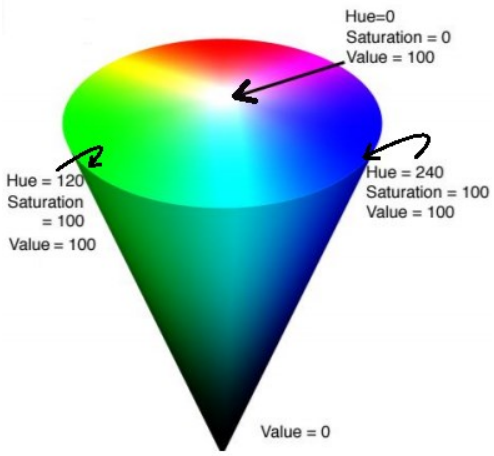


Figura 2.4. Cone do espaço HSV (Adaptado de CARDANI, 2007)

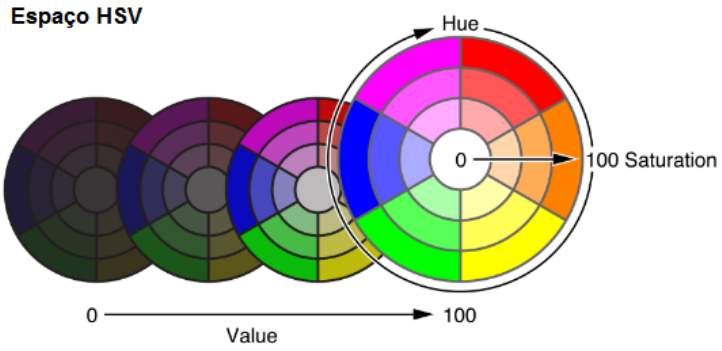


Figura 2.5. Vista com realce das seções circulares do espaço HSV (APPLE, 2005)

2.1.2 Redução de ruído e operações morfológicas

Imagens digitais são produzidas por diversas etapas durante a aquisição através de uma câmera eletrônica. Essas etapas englobam desde a irradiação (densidade de energia solar incidente, por unidade de tempo, numa determinada superfície) da luz visível na câmera, o conjunto ótico das lentes, abertura das lentes, abertura do obturador, conversão fotoelétrica do sensor (*CCD – Charge-Coupled Device*, ou *CMOS – Complementary metal-oxide-semiconductor*), ganho de sinal e conversão analógico-digital, até as etapas de pós-processamento da imagem, como a compressão. Durante todo esse processo de aquisição, ruído é adicionado à informação (imagem), ou seja, um sinal que distorce a informação. Em especial, as etapas que envolvem o *chip* do *sensor* são as que mais inserem ruído no processo, como as aproximações nos valores do sinal analógico adquirido (luz) ocasionadas

pela conversão analógico-digital (quantização). Além disso, as etapas de ganho de sinal e integração do sinal durante o tempo de exposição do *sensor* à luz capturada, sendo esta última fortemente influenciada pelo tempo de abertura do obturador, também podem amplificar ou somar ruído ao sistema (SZELISKI, 2010). Dessa forma, com ruído no sinal adquirido, os valores de intensidade produzidos pelo *sensor* podem se diferenciar para mais ou para menos dos valores originais das cores observadas. A Figura 2.6 retrata algumas das etapas mais comuns durante a aquisição de uma imagem digital, conforme delineado neste parágrafo.

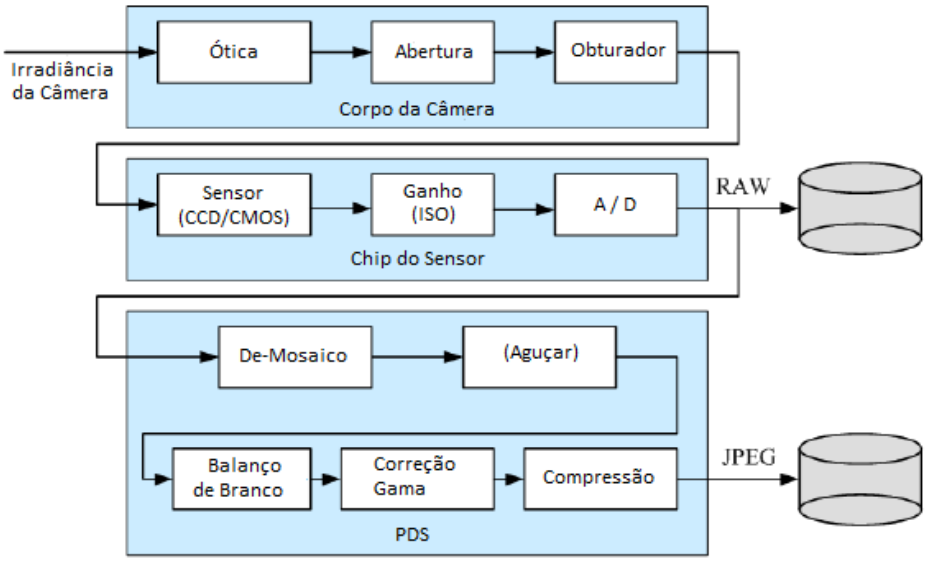


Figura 2.6. Processo de aquisição de uma imagem digital, apresentando as diversas fontes de ruído incluídas no processo, bem como etapas típicas de pós-processamento de imagens (Adaptado e Traduzido de SZELISKI, 2010)

Portanto, para o correto funcionamento de uma aplicação baseada em filtragem de cor, ou segmentação, ou qualquer processo que busque identificar, ou selecionar, contornos em uma imagem, faz-se necessária uma atenuação destes ruídos.

2.1.2.1 O filtro gaussiano

O ruído pode ser causado por diferentes fontes ao longo do processo de aquisição e, geralmente estas fontes de ruído possuem caráter aleatório, produzindo flutuações aleatórias no sinal da imagem. Com frequência, o ruído em imagens digitais é classificado como do tipo aditivo, referido como ruído branco gaussiano. Esse tipo de ruído é caracterizado por possuir componentes em toda banda de frequência, com intensidade uniforme em todos os pontos e

distribuição de probabilidade gaussiana (VISWANATHAN; KRISHNAN, 2013). Assim, a filtragem gaussiana é usada neste trabalho com o intuito de minimizar parte do ruído presente nas imagens adquiridas.

O filtro gaussiano é um filtro espacial do tipo linear (também chamado de convolução, devido a operação matemática realizada nesse tipo de filtro), mais especificamente um filtro do tipo passa-baixa (INSTRUMENTS, NATIONAL, 2011). Segundo Corke (2011), a operação de convolução é uma das bases da área de processamento digital de imagens, e pode ser utilizada em diversos tipos de processamento, linear e não-linear, realizando tarefas de cálculo de gradiente à detecção de contorno. A operação de convolução é descrita pela Eq. (2.1):

$$O[u,v] = \sum_{(i,j)\in W} I[u+i, v+j]K[i,j], \qquad \forall\,(u,v) \in I \tag{2.1}$$

onde,

- O é a imagem de saída;
- I é a imagem de entrada;
- K é o *kernel* (ou filtro) de convolução, sendo $K \in \mathbb{R}^{wxw}$;
- W é a janela do *kernel*;
- $w\ x\ w$ uma região retangular com lado $w = 2h + 1$;
- $h \in \mathbb{Z}^+$ é a "metade do comprimento ou lado";
- i e j são as coordenadas da matriz da janela do *kernel*, com $i, j \in [-h, h]$;
- u e v são as coordenadas horizontal e vertical da matriz da imagem, e limitadas pela largura e comprimento da imagem, respectivamente.

Como observado na Eq. (2.1), a operação de convolução transforma cada *pixel* da imagem de entrada, representado pelo par de coordenadas (u, v), no *pixel* da imagem de saída na mesma posição (u, v), onde o *pixel* de saída é o resultado da soma entre a multiplicação de cada *pixel* da imagem de entrada com o respectivo valor de mesma posição no *kernel*, ou seja, uma operação *pixel* a *pixel* entre os *pixels* da imagem de entrada e os valores correspondentes no *kernel*. Acrescenta-se ainda que o *kernel* possui sua coordenada

$(i,j) = (0,0)$ no centro do *kernel*. Um resumo visual da operação de convolução é apresentado na Fig. 2.7.

Desse modo, o filtro (ou *kernel*) gaussiano bidimensional pode assim ser calculado a partir da Eq. (2.2), e retratado em sua forma matricial através da Fig. 2.8.

$$G(i,j) = \frac{1}{2\pi\sigma^2} e^{\frac{-(i^2+j^2)}{2\sigma^2}} \tag{2.2}$$

onde $G(i,j)$ é o valor do filtro para as coordenadas espaciais (i,j) com média zero e desvio padrão σ (FERGUS, 2016).

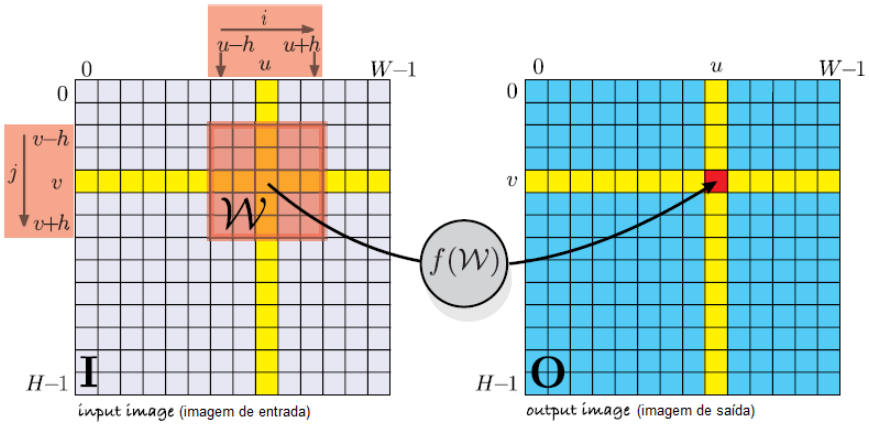


Figura 2.7. Operação de processamento espacial (convolução). A região sombreada vermelha apresenta a janela W do *kernel* (CORKE, 2011)

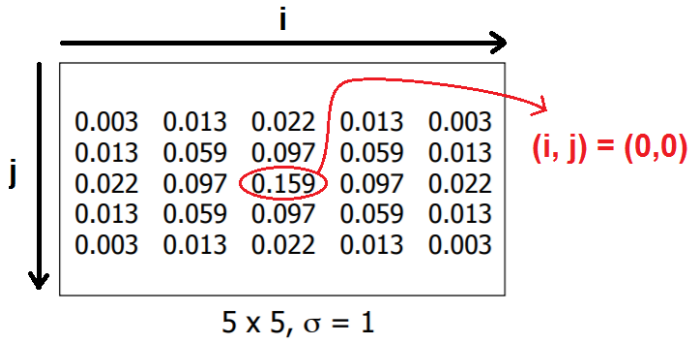


Figura 2.8. Matriz do *kernel* gaussiano de tamanho 5x5 e desvio padrão $\sigma = 1$ (Adaptado de FERGUS, 2016)

O filtro gaussiano pode ser entendido com maior facilidade se observado visualmente através do gráfico da Eq. (2.2) ilustrado na Fig. 2.9, e do seu efeito em uma imagem ligeiramente serrilhada, como apresentado na Fig. 2.10.

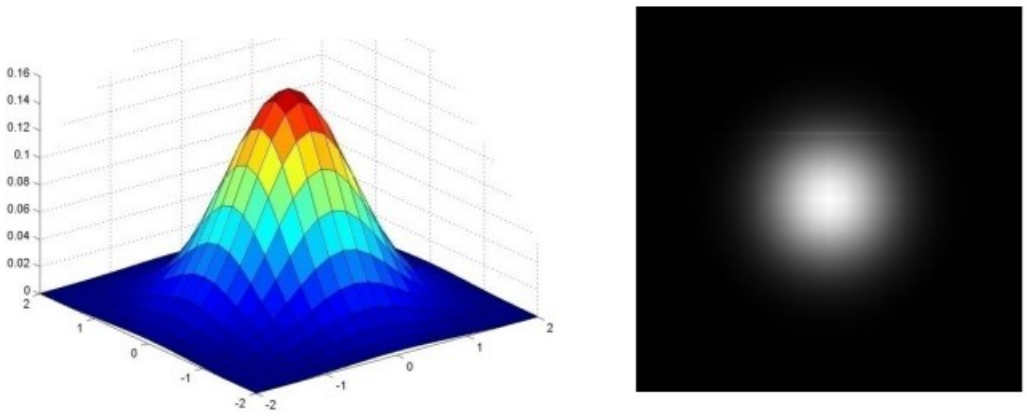


Figura 2.9. Gráfico da equação do filtro gaussiano e seu efeito em uma imagem em escala de cinza. Nota-se a maior intensidade no centro da distribuição de probabilidade e sua diminuição conforme se aproxima da borda (FERGUS, 2016)

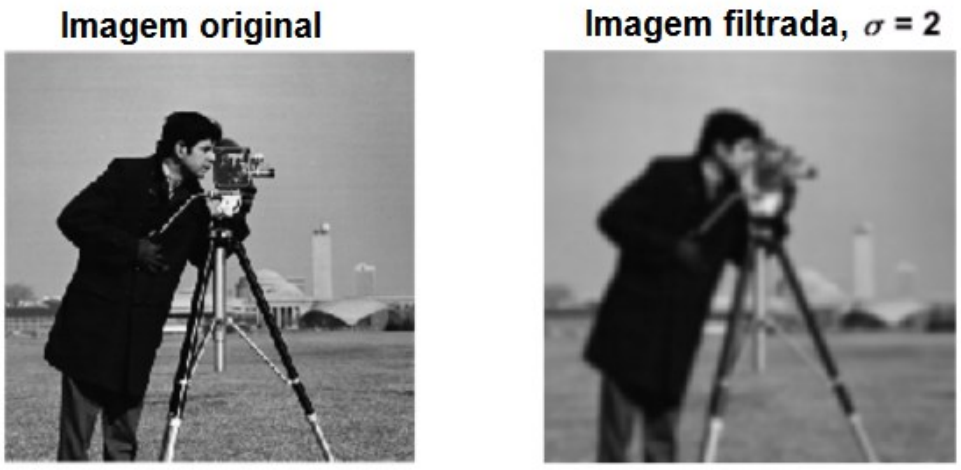


Figura 2.10. Efeito do filtro gaussiano em imagem ligeiramente serrilhada para um desvio padrão $\sigma = 2$ (Adaptado e Traduzido de MATHWORKS, 2017)

2.1.2.2 Erosão e dilatação

A filtragem de cor geralmente gera um imagem binária, ou seja, uma imagem onde todos os *pixels* podem assumir apenas dois valores de intensidade, comumente apresentada utilizando as cores branca e preta, os extremos da representação de intensidade. Assim, em uma imagem binária, na filtragem de cor vermelha, por exemplo, todos os pontos em vermelho assumem a cor branca e todos os demais pontos assumem a cor preta, com o intuito de se distinguir facilmente a região em vermelho. O valor de intensidade zero é atribuído ao preto e o valor um, ou 255, ao branco, como apresentado na Fig. 2.11 (FLUMINENSE, 2017).

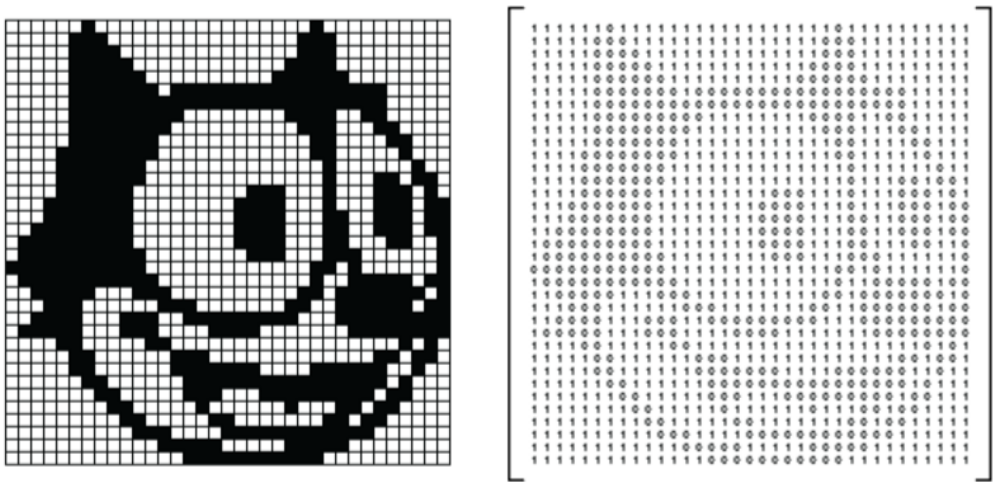


Figura 2.11. Exemplo de imagem binária (FLUMINENSE, 2017)

Contudo, ruídos com matiz próxima à cor filtrada podem gerar falsos positivos nas imagens e a mesma se tornar granulada em alguns pontos da imagem binária resultante. Uma forma de atenuar estes ruídos é utilizando o filtro gaussiano já explicado. Porém, existe um conjunto de operações, chamadas de operações morfológicas, que produzem ótimos resultados na eliminação desses pontos granulados.

O termo morfológico denota a ação de lidar com a forma e estrutura de animais e plantas, um ramo da biologia. No contexto matemático, a representação morfológica é extremamente útil em lidar com a descrição de formas e contornos (WOODS; GONZALEZ, 2008). As operações morfológicas são as operações mais comuns em imagens binárias. Essas operações geralmente realizam a convolução da imagem de entrada com um elemento estruturante (como um *kernel* ou até mesmo uma forma mais complexa), gerando uma nova imagem ou apenas descritores, como o contorno de um objeto (SZELISKI, 2010).

Ainda segundo Szeliski (2010), as operações morfológicas mais utilizadas são: dilatação, erosão, abertura (uma dilatação após uma erosão) e fechamento (uma erosão após uma dilatação). Essas quatro operações são bem intuitivas, como a abertura e o fechamento, e podem ser melhor descritas através de uma ilustração, como a apresentada na Fig. 2.12, onde uma elemento estruturante de tamanho 5x5 foi utilizado para realização das operações morfológicas descritas na figura. Dessa forma, operações como a abertura podem ser usadas para erodir granulados em imagens filtradas, criando um efeito de atenuação sobre eles ou mesmo eliminando-os, e então expandir, ou dilatar, partes da imagem filtrada desejadas que estão ligeiramente dispersas, de forma a uni-las.

Algumas dessas operações morfológicas são utilizadas neste trabalho e serão citadas no capítulo referente à aplicação *Android* desenvolvida. Contudo, por se tratarem de operações não-lineares, o seu tratamento matemático está fora do escopo deste trabalho, mas pode ser encontrado na literatura correspondente, como em Shapiro e Haralick (1992, Seção 5.2).

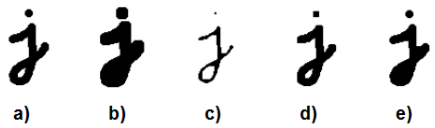


Figura 2.12. Operações morfológicas mais utilizadas. a) imagem original; b) dilatação; c) erosão; d) abertura; e) fechamento (SZELISKI, 2010).

A Figura 2.13 apresenta mais uma aplicação das operações morfológicas. Nesta figura elas são utilizadas no processamento e leitura da biometria da digital humana após binarização da imagem.

2.1.3 Identificação de contornos

Após a binarização de uma imagem, o próximo passo é frequentemente identificar contornos presentes nessa imagem, com o intuito de discernir diferentes objetos, por vezes, objetos de geometria simples, como um retângulo. De acordo com KAEHLER e BRADSKI (2011), um contorno é basicamente uma lista, que pode ser uma lista simples de pontos no espaço bidimensional da imagem ou uma lista de formas de representação complexa, como a cadeia de *Freeman* (GONZAGA, 2017). Em todos os casos, essa lista tem o intuito de representar um traço ou uma curva na imagem.

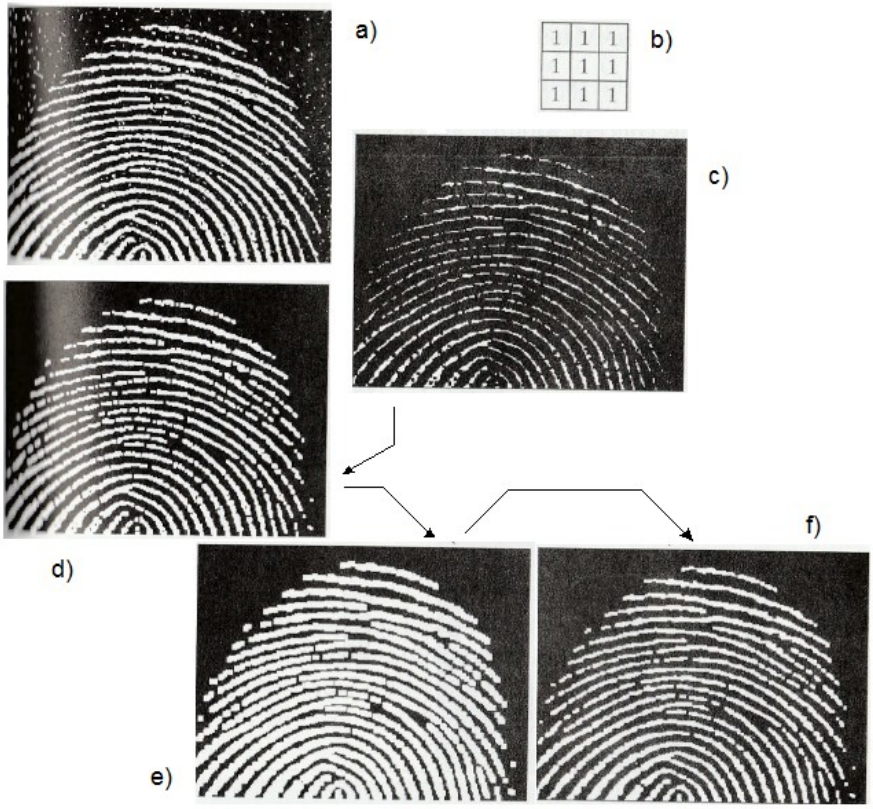


Figura 2.13. Operações morfológicas em imagem binária da digital humana. a) imagem com ruído; b) elemento estruturante; c) imagem erodida; d) abertura da imagem; e) dilatação da abertura; f) fechamento da abertura (Adaptado de WOODS; GONZALEZ, 2008 *apud* Instituto Nacional de Padrões e Tecnologia – NIST)

A biblioteca *OpenCV* contém funções otimizadas que permitem a busca e o retorno de contornos a partir de uma imagem, sendo estas utilizadas pela aplicação desenvolvida neste trabalho.

2.1.4 A Transformada circular de Hough

Frequentemente é necessário detectar objetos com formas geométricas simples em uma imagem, uma tarefa que geralmente pode ser realizada com facilidade e sem problemas por um algoritmo de identificação de contorno. Contudo, dependendo da forma geométrica a ser detectada, métodos alternativos específicos podem ser utilizados. Um dos métodos alternativos é a Transformada de *Hough*, que pode ser generalizada para identificar linhas,

círculos, elipses e outros objetos de formas geométricas arbitrárias, sendo uma ferramenta padrão na área de processamento digital de imagens. Cabe ainda dizer que a Transformada de *Hough* foi inventada na *International Business Machines* (*IBM*) por Paul Hough, recebendo o nome do seu inventor, em 1962, e ainda é muito utilizada nos dias atuais.

Neste trabalho, devido à necessidade de se identificar o robô *E-Puck* nas imagens capturadas pelo dispositivo *Android*, sendo este robô de formato circular, uma variação da Transformada de *Hough* para o caso circular é usada. A Transformada de Hough, ou neste caso, a Transformada circular de *Hough*, assim chamada, é capaz de produzir bons resultados mesmo em situações em que os contornos presentes nas imagens estejam desconectados ou unidos à pequenos contornos aleatórios, na maioria dos casos devido a oclusões causadas por outros objetos ou por ruído não filtrado (BALLARD, 1981).

Internamente, algoritmos que realizam a Transformada circular de *Hough* aplicam, em geral, alguma técnica de segmentação de imagens em primeira instância. A segmentação é responsável por decompor uma imagem em partes, formando regiões que representam a imagem na forma de componentes conectados, como contornos, arestas, ou até mesmo formas geométricas simples, como um retângulo ou um círculo. Não existe um método único para realizar a segmentação de uma imagem, porém, alguns métodos têm sua aplicação consagrada pela capacidade de segmentar uma imagem genérica, e identificar uma quantidade satisfatória de arestas presentes na imagem, produzindo resultados significantes mesmo na presença de ruídos. Um desses métodos é o detector de bordas de *Canny* (KAEHLER; BRADSKI, 2017).

O detector de bordas de *Canny* é o aperfeiçoamento de um filtro comum, e bastante popular, utilizado em segmentação de imagens, o filtro Laplaciano. Este aperfeiçoamento do filtro Laplaciano foi realizado por J. Canny em 1986, e recebeu seu nome. O detector de bordas de *Canny* trabalha com derivadas da imagem em relação às coordenadas espaciais da mesma, combinando estas derivadas em derivadas direcionais que informam os pontos de máxima ao longo da imagem. Estes pontos de máxima são possíveis candidatos na formação de arestas (ou bordas), sendo posteriormente tratados por um filtro de histerese, onde valores limítrofes, máximo e mínimo, são aplicados à estes pontos para determinar se estes são aceitos de fato como pontos pertencentes à uma aresta (KAEHLER; BRADSKI, 2017). Cabe ressaltar que o detector de bordas utiliza imagens em escala de cinza para o processo de segmentação, e geralmente, um filtro gaussiano e aplicado à esta imagem para reduzir as chances de identificação de uma falsa aresta por parte do detector. Por fim, com a imagem segmentada, algumas técnicas podem ser empregadas de forma a averiguar se as bordas (ou arestas) produzidas formam algum contorno ou forma geométrica em específico.

Segundo Rhody (2005a), a Transformada circular de *Hough* emprega um conjunto de equações paramétricas dadas pelas Equações (2.3) e (2.4) para identificação de círculos a partir das arestas produzidas no algoritmo de detecção de bordas. Essas equações são as usadas para descrever um círculo de raio R e centro (a, b).

$$u = a + Rcos(\theta) \tag{2.3}$$

$$v = b + Rsen(\theta) \tag{2.4}$$

onde $0 \leq \theta \leq 2\pi$.

Assim, com base nas Equações (2.3) e (2.4), todo o conjunto de pontos (u, v) do perímetro de um círculo é percorrido quando o ângulo θ varia de 0 à 2π, devendo todos os *pixels* em uma imagem, que estejam no perímetro de um círculo qualquer de raio R e centro (a, b), obedecerem à essa relação de equações paramétricas.

Deste modo, o papel da Transformada de *Hough* é encontrar os três parâmetros (a, b, R) representantes de cada círculo formado por um conjunto de *pixels* qualquer, sendo, portanto, um problema de busca em um espaço tridimensional. Contudo, este problema pode ser dividido em dois casos gerais: círculos com raio conhecido e círculos com raio desconhecido.

No caso de círculos com raio conhecido, é necessário encontrar apenas os parâmetros (a, b), reduzindo o problema à uma busca em um espaço bidimensional, e reduzindo também o custo computacional. Este caso é ilustrado na Fig. 2.14.

Como observado na Figura 2.14, para cada ponto (u, v), pertencente ao círculo no espaço geométrico, há um círculo correspondente de centro (u, v) e raio R no espaço paramétrico $a - b$. Nota-se também que para cada ponto (u, v) projetado no espaço paramétrico, o círculo correspondente formado se intersecciona com os outros círculos formados pelos outros pontos (u, v) do espaço geométrico.

Esta intersecção dos círculos no espaço paramétrico é o ponto (a, b) que indica o possível centro do círculo no espaço geométrico que engloba todos os pontos (u, v). Essa decisão do círculo é feita por um quantizador, ou seja, quanto mais círculos se

interseccionarem no espaço paramétrico $a-b$ no ponto (a,b), maior a chance dos pontos (u,v) no espaço geométrico formarem um círculo de raio R e centro (a,b).

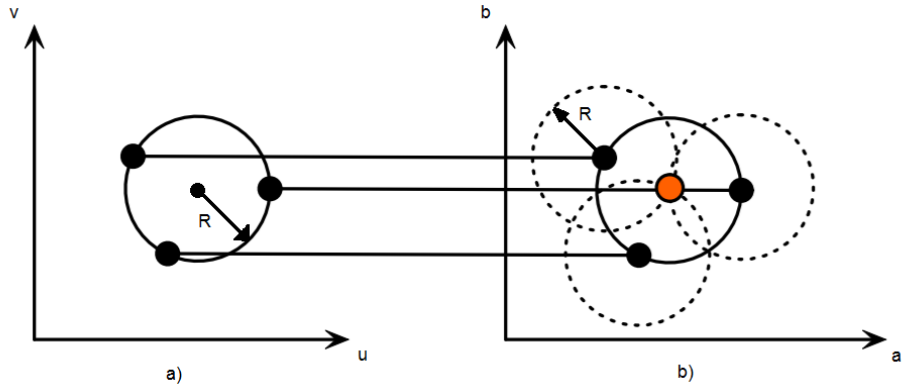


Figura 2.14. a) círculo no espaço geométrico $u-v$; b) círculo no espaço paramétrico $a-b$ (Adaptado de Rhody, 2005a)

O quantizador é realizado por meio do chamado, vetor acumulador, no algoritmo da Transformada de *Hough*. De acordo com Rhody (2005b), o espaço paramétrico $a-b$ é dividido em células com valor inicial zero, como as células em um vetor, e para cada ocorrência de uma intersecção em um ponto (a,b) qualquer, é somado (ou acumulado) o valor um ao valor da célula contendo este ponto (a,b). Ao final do processamento da imagem, contabiliza-se todas as células do vetor acumulador, e a partir de um valor limítrofe pré-estabelecido, todas as células que tiverem valor maior que este valor limítrofe, possuem possíveis círculos de centro (a,b) correspondentes no espaço geométrico. Esse processo de quantização é às vezes chamado de votação.

Há ainda o caso em que na busca por círculos o raio R não é conhecido. Neste caso, o problema é de busca em um espaço tridimensional, como já mencionado. Além disso, este é o caso mais comum, já que geralmente não se sabe de antemão o valor do raio do(s) círculo(s) presente(s) na imagem. A Figura 2.15 ilustra o espaço paramétrico tridimensional na forma de um cone.

Ainda segundo Rhody (2005a), na busca tridimensional a abordagem é a mesma, a diferença é que cada ponto (u,v) no espaço geométrico será representado por uma superfície cônica no espaço paramétrico, como a apresentada na Fig. 2.15. O vetor acumulador passa a ser uma matriz tridimensional e a busca no espaço paramétrico é realizada através da

intersecção do maior número de superfícies cônicas. Geralmente, os algoritmos que realizam a transformada de *Hough*, como o da função correspondente na biblioteca *OpenCV*, solicitam a informação dos raios mínimo e máximo pelo os quais buscar no espaço paramétrico, reduzindo a superfície cônica à proporções finitas.

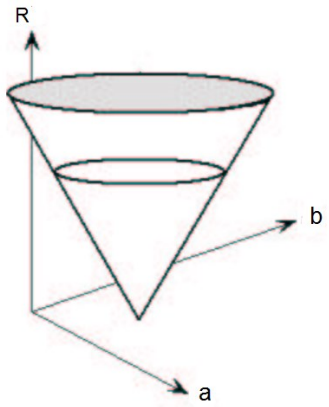


Figura 2.15. Espaço paramétrico $a - b - R$ tridimensional (Adaptado de Rhody, 2005a)

2.2 O sistema operacional *Android*

O ano de 2017 trouxe uma reviravolta no mercado de sistemas operacionais utilizados no mundo, até então dominado pelo *Windows* da empresa *Microsoft*. Segundos dados do *StatCounter*, uma empresa responsável pela análise de utilização de serviços da internet por sistema operacional (SO), o sistema operacional *Android* conta com 37,93% de todos os usuários com acesso à internet, enquanto o *Windows* conta com a fatia de 37,91% de usuários, uma margem pequena, mas bastante significativa (ZURIARRAIN, 2017).

No Brasil, o mercado de sistemas operacionais para dispositivos móveis, como os *smartphones*, é dominado principalmente pelos sistemas *Android*, *Windows Phone* e *iOS*. Contudo, segundo dados da consultoria IDC, o *Android* é o SO utilizado em 95,5% de todos os aparelhos móveis comercializados entre Julho e Setembro de 2016, ou seja, dos 11,1 milhões de *smartphones* vendidos neste período (HIGA, 2016). Devido à essa disseminação dos *smartphones* e do sistema operacional *Android* embarcado nestes aparelhos, o *Android* é o SO escolhido para este trabalho.

Todas as versões do SO *Android* até a 7.0 possuem nome de doces, e em ordem alfabética. Como será apresentado no capítulo VI, a aplicação desenvolvida foi testada na versão 4.4.2 do *Android*, denominada *KitKat*.

Tecnicamente, o sistema operacional *Android* é baseado no sistema operacional *Linux*, permitindo sua utilização em um variedade de dispositivos de diferentes tamanhos e poder computacional, além de ser de código aberto (*open source*). A arquitetura da plataforma *Android* fornece ainda diversas funcionalidades e recursos poderosos, acessíveis através de uma *API (Application Programming Interface) JAVA*, ou seja, um conjunto de bibliotecas escritas na linguagem de programação *JAVA*. Esta *API* se encontra no *kit* de desenvolvimento de *software* (*SDK*, do inglês *Standard Development Kit*) que acompanha o ambiente de desenvolvimento *Android Studio*, sugerido pela empresa *Google*. Assim, utilizando se a linguagem *JAVA* é possível acessar todas as funcionalidades do SO *Android*, dentre elas: criação e gerenciamento do sistema de visualização da aplicação (sistema de *views*), dos provedores de conteúdo (para troca de dados entre telas das aplicações), dos recursos de notificação e telefonia, e dos recursos de *hardware*, como a câmera do dispositivo, adaptadores *Bluetooth* e *WiFi*, acelerômetros, bússola e vários outros. O SO *Android* conta ainda com sistemas flexíveis de compilação baseados no *Gradle* (GRADLE INC, 2017) e no *CMake* (este último para código nativo) (KITWARE, 2017). É possível ainda a programação de aplicativos *Android* utilizando-se código nativo C/C++. Para isso, além do *SDK*, um *kit* de desenvolvimento nativo (*NDK*, do inglês Native Development Kit) deve ser utilizado, e em geral, juntamente com a *JNI* (*Java Native Interface*) e o sistema de compilação *CMake*. Estes *softwares* criam a ponte entre o código *JAVA* e o código C/C++.

Cabe ressaltar ainda que toda aplicação *Android* é desenvolvida a partir de um projeto. Um projeto define um nome único para a aplicação e permite a escolha da versão mínima do SO *Android* a ser suportada pela aplicação. Além disso, a aplicação desenvolvida com base em um projeto é construída utilizando-se blocos de construção, chamados de componentes de aplicativos, onde cada componente representa um meio de entrada do sistema operacional no aplicativo. Cada componente possui seu próprio ciclo de vida, ou seja, a forma pela qual o componente é criado e destruído. Existem no total quatro tipos de componentes com finalidades distintas: atividades (do inglês, *activities*), serviços (do inglês, *services*), provedores de conteúdo (do inglês, *content providers*) e receptores de transmissão (do inglês, *broadcast receivers*). Neste trabalho, os componentes, atividade, provedor de conteúdo e receptor de transmissão, foram utilizados. Foi utilizado também o recurso de *fragmentos* do SO *Android* (do inglês, *Fragments*). Por fim, não cabe a este trabalho apresentar uma lista extensa e explicativa das funcionalidades presentes no SO *Android*, sendo estas

funcionalidades encontradas em detalhes e em forma de guias de utilização na referência em ANDROID (2017). Contudo, uma breve explicação da integração entre a aplicação *Android* e a biblioteca *OpenCV* é fornecida no capítulo IV devido à sua relevância neste trabalho.

CAPÍTULO III

TRABALHOS CORRELATOS

Este capítulo apresenta técnicas e abordagens adotadas em outros projetos que se relacionam com o tema deste trabalho. São apresentados também os resultados obtidos pelos autores destes trabalhos de acordo com sua relevância neste contexto.

3.1 Ambiente de testes e técnicas utilizadas

Diversas configurações têm sido empregadas na literatura para construção de um ambiente de testes para estudos de planejamento de trajetória em robótica móvel. Estes estudos geralmente envolvem a utilização de técnicas como grafos ou autômatos celulares.

Sumbal (2016) desenvolveu um campo com uma caixa de madeira onde os obstáculos e um robô *E-Puck* foram posicionados. Foi proposto o desenvolvimento de um programa para identificação dos objetos em campo, além do planejamento de trajetória do robô *E-Puck*, de modo que este saísse de sua posição, inicialmente aleatória, e atingisse um objetivo especificado pelo usuário sem colidir com nenhum obstáculo, seguindo pela menor trajetória possível. Para isso uma câmera *Sony SSCDC198P* foi posicionada acima do campo, e os objetos no campo foram marcados em diferentes cores, sendo: magenta e verde claro para o robô, verde claro para os obstáculos, e um verde mais escuro para as bordas do campo. Foram utilizadas duas cores no robô para identificação da sua orientação. Na Figura 3.1 são ilustrados os componentes utilizados na configuração adotada por Sumbal (2016).

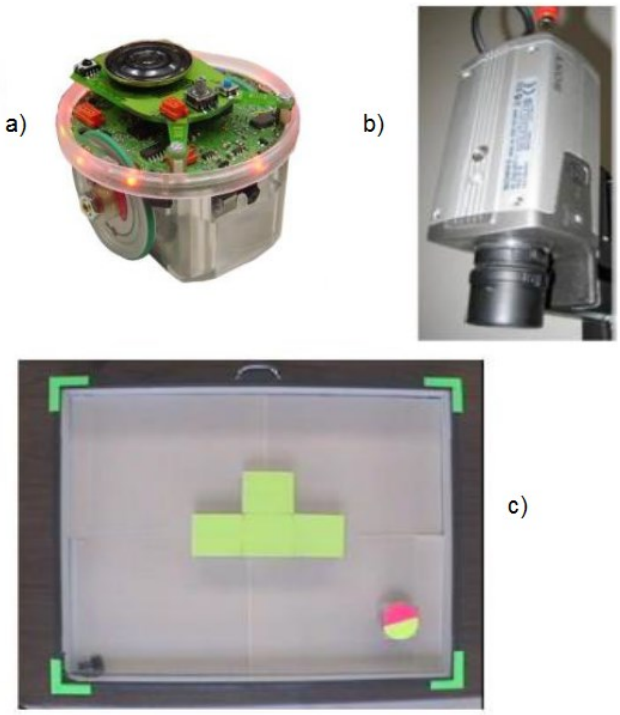


Figura 3.1. a) robô *E-Puck*; b) câmera Sony para captura das imagens; c) campo com objetos (Adaptado de SUMBAL, 2016)

A câmera capturava as imagens em *RGB* e um algoritmo de conversão de espaço de cor convertia a imagem para o espaço *HSV*. No espaço *HSV* os objetos foram filtrados de acordo com suas cores, e após identificação, foi atribuída uma nova cor para cada objeto, onde o verde claro passou a ser marrom, o magenta passou a ser azul claro e o verde mais escuro se tornou amarelo. A partir deste ponto um algoritmo calculou os momentos de área para as bordas em amarelo e para a metade azul claro, encontrando os centroides destes objetos e definindo a posição central das bordas e do ponto inicial de partida do robô. Nos pontos centrais identificados foi atribuído um ponto em vermelho. Algoritmos de erosão e dilatação foram aplicados aos objetos filtrados para fechamento dos espaços vazios e correta identificação dos mesmos. No caso dos obstáculos, foi aplicada uma função denominada envoltória convexa (do inglês, *Convex Hull*) para definição do menor conjunto de pontos envolvendo os obstáculos. Essa operação foi necessária para o planejamento de trajetória que trabalha neste caso a partir de uma técnica baseada em grafos de visibilidade. Assim, todos os cantos dos obstáculos foram tratados como nós dos grafos, e para isso um outro algoritmo de identificação de contornos e bordas foi utilizado, o detector de bordas de *Canny* (do inglês *Canny edge detector*). A partir da identificação das bordas, seus vértices foram retornados e inseridos no grafo de visibilidade. Após todo o processo, Sumbal (2016) aplicou

algoritmos para identificação do menor caminho a ser percorrido no grafo. A Figura 3.2 ilustra todos os passos mencionados e os seus resultados.

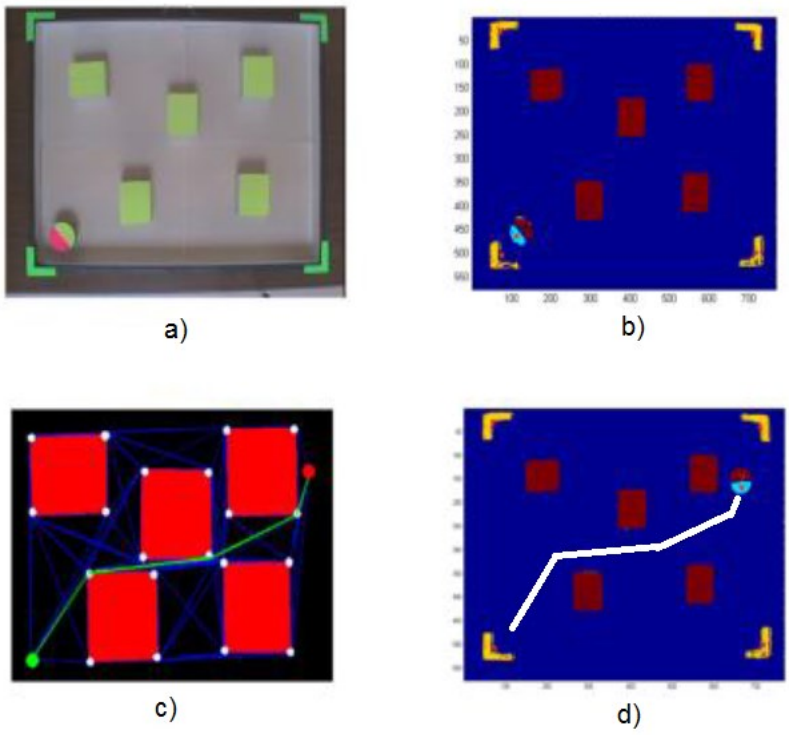


Figura 3.2. a) imagem original capturada; b) imagem após identificação dos objetos; c) grafo de visibilidade e menor caminho definido pelo algoritmo A* a partir do ponto inicial em verde e o final em vermelho; d) caminho seguido pelo robô (Adaptado de SUMBAL, 2016)

Observa-se na Figura 3.2c que cada vértice de cada obstáculo constituiu um nó no grafo e o algoritmo realiza a busca pelo menor trajeto em cada obstáculo e posteriormente em todos os grafos, até que o menor trajeto do ponto inicial ao objetivo fosse calculado. Os testes mostraram que as condições de iluminação são um fator de influência na segmentação da imagem (identificação dos objetos). Além disso, quanto maior a distância do objetivo em relação ao ponto inicial, maior o tempo de processamento do algoritmo de busca para se encontrar o menor trajeto dentre a maior quantidade de grafos. Sumbal (2016) observou também que uma técnica baseada em simplificação por polígonos seria mais eficiente na definição do conjunto mínimo de pontos para definição dos obstáculos.

Em um segundo trabalho realizado por Syed e Kunwar (2014) foi utilizada uma abordagem baseada em autômatos celulares e uma configuração com marcadores circulares sobre os objetos em um campo de cor azul uniforme para fins de identificação dos objetos. Para captura das imagens do campo a câmera *RGB-D* presente no produto *Kinect* da empresa *Microsoft* foi utilizada. Após captura das imagens pelo *Kinect* um algoritmo baseado na Transformada circular de *Hough* foi responsável pela identificação dos marcadores no objetos, seguindo para um pós-processamento baseado em cor, onde as cores branca, vermelha e verde representavam, respectivamente, o robô, obstáculos dinâmicos e obstáculos estáticos. No caso do robô, dois círculos brancos foram usados, um maior para o centro do robô e um menor em sua extremidade para identificação da orientação do mesmo. Com todos os objetos identificados, um espaço de trabalho foi construído. Este espaço de trabalho foi convertido no chamado espaço de configuração, onde levando-se em conta que o robô possa possuir qualquer forma arbitrária, uma transformação do perfil ou forma do robô é realizada e este passa a ser representado como uma forma pontual no espaço de configuração. Além disso, houveram ações de expansão ou redução da forma dos obstáculos para uniformizar suas dimensões com a dimensão do robô, ações necessárias para o tratamento baseado em espaço de configuração, porém, fora do escopo de detalhe deste trabalho. Essa adequação da forma do robô e dos obstáculos foi realizada através de uma soma de *Minkowski*, formando o espaço de configuração.

Com o espaço de configuração formado, foram atribuídos todos os possíveis estados de orientação do robô à este espaço. Com esta informação, e a informação do objetivo pré-definida, foi utilizado um algoritmo baseado em autômatos celulares e relação pai-filho entre as posições dos objetos para definição da menor trajetória a ser seguida pelo robô de forma a se evitar os obstáculos e atingir o objetivo. A Figura 3.3 ilustra o experimento montado.

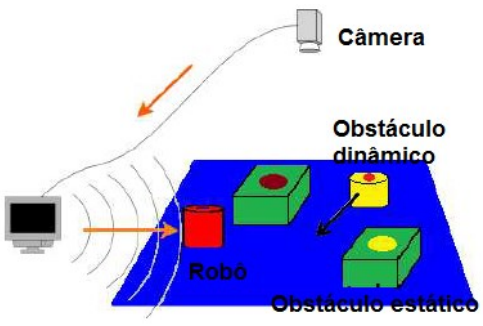


Figura 3.3. *Hardware* experimental (Adaptado e Traduzido de SYED; KUNWAR, 2014)

As etapas de configuração durante o experimento estão ilustradas na Fig. 3.4. Observa-se ainda na Fig. 3.3 que a comunicação com o robô foi realizada de forma sem fio através de um computador.

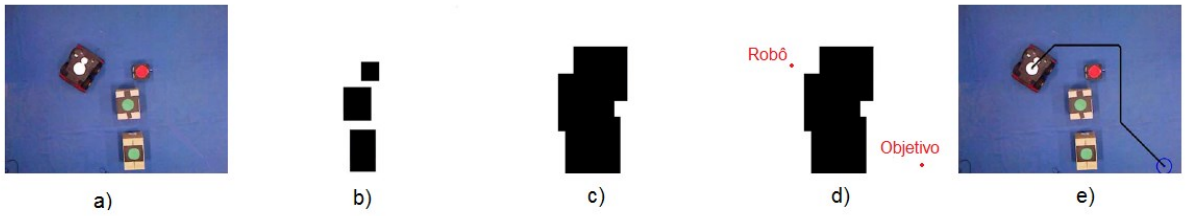


Figura 3.4. Etapas de configuração do campo de testes. a) aquisição da Imagem; b) identificação dos obstáculos; c) espaço de configuração; d) robô e objetivo identificados; e) trajeto planejado (Adaptado e Traduzido de SYED; KUNWAR, 2014)

Os testes realizados apontaram para um planejador de trajetória baseado em autômatos celulares que sempre produz um trajeto ótimo e atingi resultados em escala de tempo real quando comparado com algoritmos de planejamento de trajetória como o A*.

Outra abordagem semelhante à de Syed e Kunwar (2014) é realizada por Behring et al. (2000). Uma arena do tamanho de uma mesa padrão de tênis de mesa com um robô perseguindo uma bola foi utilizada, seguindo-se as regras de confecção de uma das competições de futebol de robôs da *Robocup*. Uma câmera digital *Connectix Color QuickCam 2* foi posicionada sobre a mesa para captura das imagens. A ideia principal foi a divisão da arena em células retangulares, onde o trajeto a ser percorrido pelo robô foi dado pela configurações de células de modo que uma célula fosse a inicial, contendo o ponto de partida do robô, e outra célula representasse o objetivo final a ser alcançado. Todas as demais células livres compartilhando um contorno comum com algumas das células mencionadas, foram consideradas células vizinhas. Assim como em Syed e Kunwar (2014) a configuração se deu baseada em um espaço de configuração. É dito que o espaço de configuração é adequado para problemas de planejamento de trajetória e tem como vantagem o fato de representar como um ponto, qualquer corpo rígido ou articulado. Neste caso, a divisão de células é discreta, cada obstáculo podendo ocupar mais de uma célula, e o robô, definido como um ponto no espaço de configuração, ocupando exatamente uma célula. Esta representação é ilustrada na Fig. 3.5.

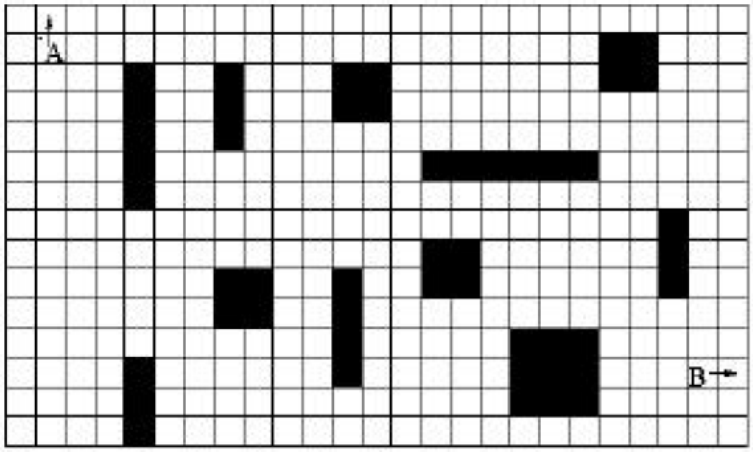


Figura 3.5. Espaço de configuração discreto. O espaço de trabalho é decomposto em um conjunto de células retangulares. Os obstáculos ocupam um certo número de células. A posição inicial do robô e do objetivo são apresentadas como os pontos A e B, respectivamente (BEHRING et al., 2000)

Em uma primeira fase um algoritmo de processamento de imagens identificou os objetos e a posição de cada um, e em seguida a imagem foi transformada em sua representação baseada em células, definindo quatro estados possíveis para cada célula: livre, obstáculo, posição inicial ou objetivo. Ao final da primeira fase, um algoritmo foi aplicado para definir as regras de transição típica da teoria de autômatos celulares, e cada obstáculo na imagem original foi expandindo para compreender a quantidade de células que o representasse, assim como realizado por Syed e Kunwar (2014), concluindo a fase dois. Em uma terceira fase o algoritmo de planejamento de trajeto para definição do menor trajeto livre de colisões foi executado. As fase um e dois estão representadas na Fig. 3.6.

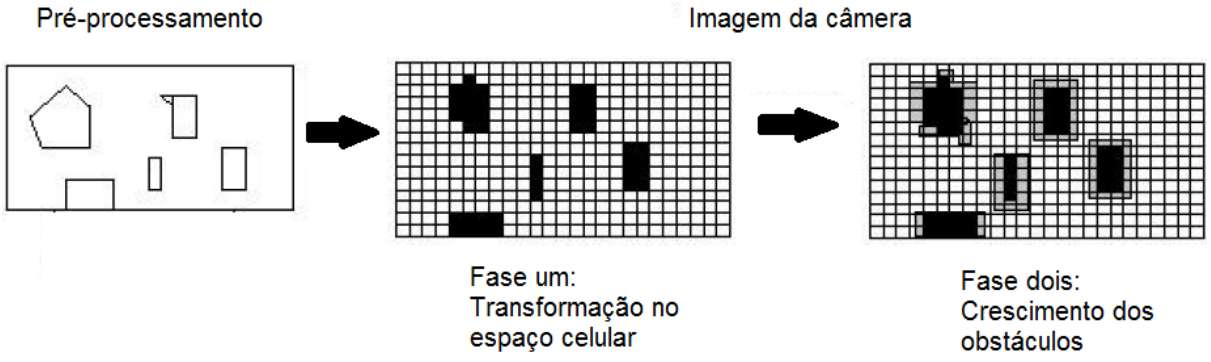


Figura 3.6. Processo de transformação da imagem em um espaço celular e crescimento dinâmico dos obstáculos (Adaptado e Traduzido de BEHRING et al., 2000)

Segundo testes realizados, a abordagem de divisão de células baseada na identificação dos objetos pela câmera produziu resultados consistentes ao longo de várias repetições e se mostrou eficiente no planejamento de trajetória em um ambiente dinâmico. Por fim, a Figura 3.7 ilustra o campo original capturado pela câmera e o resultado da identificação dos objetos, decomposição em células do campo e o caminho a ser seguido pelo robô.

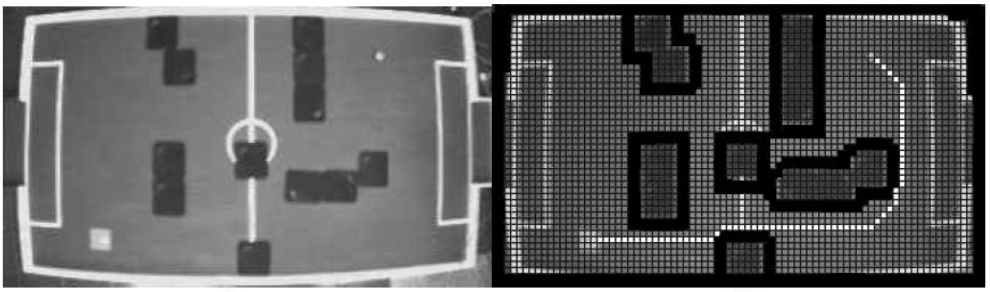


Figura 3.7. a) imagem original; b) resultado da abordagem por divisão em células, apresentando o robô, o trajeto definido em branco e os obstáculos expandidos em preto (BEHRING et al., 2000)

Um quarto ambiente de experimentação foi observado em Siano e Glielmo (2014). Este trabalho consistiu na elaboração de diversos experimentos a serem realizados por estudantes de engenharia elétrica e ciência da computação relativos à um curso de controle em tempo real. Em um dos experimentos foi proposto a implementação de um algoritmo para planejamento de trajetória utilizando um robô *E-Puck* baseando-se em algoritmos clássicos como o A* e o *Dijkstra*. O objetivo era encontrar o menor trajeto para que o robô saísse de um ponto A para um ponto B final, desviando-se dos obstáculos no caminho. Para isso foi construída uma arena com um suporte para uma câmera, onde um algoritmo de processamento de imagem realizaria a identificação do robô e dos obstáculos representados por quadrados brancos. A Figura 3.8 apresenta a configuração utilizada no trabalho destes autores.

Contudo, não foi encontrado nenhum resultado disponível de implementação deste projeto no portfólio dos autores ou da universidade em que foi aplicado, servindo apenas como uma idealização para o trabalho aqui desenvolvido. Observa-se ainda na Fig. 3.8 a divisão do campo em células, sugerindo uma possível abordagem baseada em autômatos celulares.

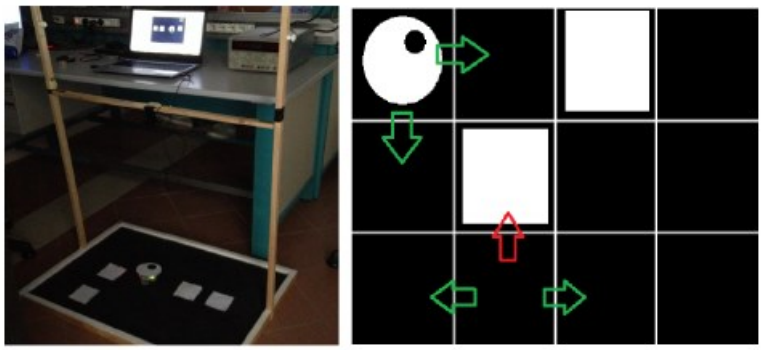


Figura 3.8. Projeto para planejamento de trajetória ótimo utilizando um robô *E-Puck* e localização por processamento de imagem (SIANO; GLIELMO, 2014)

3.2 Considerações finais

Com base nos trabalhos relacionados, pode-se destacar as técnicas e abordagens utilizadas por Sumbal (2016) e Behring et al. (2000). Sumbal (2016) propôs uma abordagem baseada na segmentação dos objetos em campo pela diferenciação de cores entre os objetos, e através do processamento da imagem por meio da transformação para o espaço de cor *HSV*. Foi proposto também a utilização de operações morfológicas, como erosão e dilatação, para fechamento dos espaços vazios durante a filtragem de cor, proporcionando melhores resultados na identificação dos objetos. Por fim, foi observado a influência da iluminação na identificação dos objetos, além de ter sido proposto a utilização de técnicas baseadas em simplificação por polígonos para definição dos obstáculos. Behring et al. (2000) por sua vez tratou da questão de divisão do campo em células para aplicação das técnicas de autômatos celulares, realizando uma abordagem baseada em espaço de configuração e alocando estados às células do campo conforme a presença, ausência ou posicionamento de objetos sobre ela. Uma abordagem mista de Sumbal (2016) e Behring et al. (2000) foi utilizada em Syed e Kunwar (2014), atingindo bons resultados, além de se destacar a identificação dos marcadores nos objetos através da Transformada circular de *Hough*.

Em todos os trabalhos mencionados foi utilizado um ambiente em formato retangular para disposição dos objetos, possuindo estes objetos, formas bem definidas, geralmente formas geométricas simples. Além disso, uma câmera superior para captura das imagens foi utilizada nestes ambientes. Houveram também relatos da utilização de uma forma de transmissão de dados sem fio para os robôs em campo em alguns dos trabalhos.

Assim, o trabalho aqui desenvolvido propõe uma abordagem mista com base nas abordagens propostas pelos demais autores, destacando se especialmente as técnicas de identificação de objetos utilizadas por Sumbal (2016), e Syed e Kunwar (2014). É utilizada também uma estratégia similar de decomposição do campo em células, como em Behring et al. (2000), contudo com uma atribuição de estados ligeiramente diferente para as células. Com relação à montagem do ambiente de testes, pode se observar uma estrutura semelhante a realizada em Siano e Glielmo (2014).

Além disso, há uma diferenciação importante com relação aos outros trabalhos. Neste trabalho, optou-se pela utilização de um dispositivo *smartphone* executando o sistema operacional *Android* como uma solução completa de *hardware*. Nos trabalhos correlatos apresentados observou-se uma configuração geralmente baseada em câmera, computador e transmissão de dados por fio ou sem fio. Ao se utilizar um *smartphone*, câmera e sistema computacional são integrados, proporcionando uma plataforma compacta e móvel, com poder de processamento suficiente para este projeto e fornecendo diversas possibilidades de transmissão de dados para o robô em campo, sem necessidade de um computador intermediário. Uma dessas opções de transmissão de dados é o protocolo de comunicação *Bluetooth*, presente no *smartphone* e suportado também pelo robô *E-Puck*, sendo este o meio de comunicação escolhido para este trabalho. Essas abordagens serão melhor apresentadas nos capítulos se seguem.

CAPÍTULO IV

A APLICAÇÃO *ANDROID*

Este capítulo apresenta a aplicação *Android* desenvolvida, denominada OCVCmakeTCCFinal. As técnicas empregadas no desenvolvimento da aplicação são apresentadas com base nos *menus* criados para cada tarefa a ser realizada pela aplicação. Assim, este capítulo detalha o modo de utilização da aplicação e dá indícios de possíveis modificações que podem ser realizadas em trabalhos futuros. A aplicação foi criada utilizando a *API Android* disponível no *SDK Android*, onde a *IDE Android Studio* foi empregada como ambiente de desenvolvimento, permitindo também a integração da aplicação com a biblioteca *OpenCV*. O *Android Studio* foi instalado no sistema operacional *Ubuntu*, baseado em *Linux.* O código fonte desta aplicação pode ser encontrado no repositório *GitHub* com *link* para *download* disponível em GlenioSP (2017). É possível também visualizar o diagrama de classes da aplicação após carregar o projeto via *IDE Android Studio*. Para isso, um *plugin* é necessário, existindo diversas opções. Um *plugin* que pode ser facilmente instalado é o *Code Iris* (JETBRAINS, 2017). Para instalá-lo, é necessário acessar o *menu File* > *Settings* > *Plugins* > *Browse Repositories* no *Android Studio* e digitar *Code Iris*, abrindo um *menu* lateral com a opção de instalação após encontra-lo. Após instalado, o *Android Studio* deve ser reiniciado e o *plugin* estará disponível para utilização via *menu View* > *Tool Windows* > *Code Iris*, permitindo que os pacotes e classes desejados sejam selecionados para visualização em um *menu* lateral que é aberto.

Por fim, para versões do sistema operacional *Android* a partir da versão 5.0, permissões de utilização da câmera devem ser dadas individualmente à aplicação para sua utilização. Essas permissões podem ser configuradas no *menu* de configurações do sistema *Android* do dispositivo, procurando-se pelo *menu* Aplicações, e em seguida escolhendo esta aplicação

após ter sido instalada. Após escolhê-la, uma janela com diversas opções de permissão será aberta para serem habilitadas ou desabilitadas.

4.1 Integração *OpenCV* e *Android*

A biblioteca *OpenCV* (*Open Source Computer Vision Library*), é uma biblioteca de código aberto (*open source*) desenvolvida com intuito de fornecer ferramentas para a solução de problemas em Visão Computacional, e que seja simples de se usar. Além disso, devido ao alto custo computacional geralmente necessário ao se empregar técnicas de visão computacional, a biblioteca *OpenCV* foi desenvolvida com foco na otimização dos algoritmos implementados por ela e na reutilização de código. Atualmente, é uma das bibliotecas de visão computacional mais populares e utilizadas no mundo. Centros de pesquisas em várias universidades à grandes empresas, como *Intel*, *IBM* e *Microsoft*, utilizam a biblioteca *OpenCV* em alguns de seus projetos. Para isso, a biblioteca *OpenCV* conta com mais de 500 funções aplicáveis à diversas áreas, desde processamento de imagens médicas à calibração de câmera, visão estereoscópica (com mais de uma câmera) e robótica. A biblioteca conta ainda com um módulo de funções de aprendizagem de máquina, pois diversas técnicas da área de visão computacional utilizam-se de técnicas de inteligência artificial (IA).

Além disso, devido à popularização da biblioteca *OpenCV*, escrita originalmente em C/C++, conta ainda com suporte à vários sistemas operacionais, como o *Windows* da *Microsoft*, as diversas versões de *Linux*, o *OS X* e o *iOS* da *Apple*, e o *Android* da *Google*. Não obstante, devido à esta variedade de plataformas suportadas, há versões da biblioteca *OpenCV* para várias das linguagens de programação mais populares, como *JAVA*, *MATLAB* e *Python*, com *binding* (vinculação de nomes) para o código em C/C++ original (KAEHLER; BRADSKI, 2017).

Neste trabalho, a versão da biblioteca *OpenCV* para o *Android* foi utilizada. Esta versão é distribuída na forma de um módulo, que pode ser importado dentro da aplicação *Android* de forma bem simples. O módulo *OpenCV* para *Android*, chamado *OpenCv4Android*, pode ser encontrado para *download* em OPENCV (2017a).

Primeiramente, com a *IDE Android Studio* e a *SDK Android* instaladas, faz-se necessário a instalação do *NDK* para execução da aplicação com código nativo. O *NDK* pode ser encontrado através do *menu Tools* > *Android* > *SDK Manager* ou na barra de ferramentas

presente na *IDE Android Studio*, na versão em inglês. Com o *SDK Manager* aberto, pode se navegar até a aba *SDK Tools* e o *NDK* estará listado para instalação. Além do *NDK*, é necessário que o *CMake* e o *LLDB* (para depuração do código) sejam instalados também. Assim, com a *IDE Android Studio*, o *SDK Android*, o *NDK*, o *CMake* e o *LLDB* instalados, o passo a passo para a criação de um novo projeto *Android* com suporte para código nativo C/C++ e integração com a biblioteca *OpenCV* pode ser encontrado em Leadrien (2017). O guia de configuração apresentado em Leadrien (2017), bem como o código de exemplo *Tutorial3-CameraControl* fornecido com o módulo *OpenCv4Android* e encontrado no diretório *samples*, foram utilizados como base para a aplicação desenvolvida aqui neste trabalho. A versão do *OpenCv4Android* utilizada foi a 3.2.0.

Cabe ressaltar que o módulo *OpenCv4Android* permite também que a aplicação *Android* possa utilizar as funções da biblioteca *OpenCV* implementadas em *JAVA*, como um *binding* para a linguagem C/C++. A grande maioria das funções C/C++ da biblioteca *OpenCV* possuem correspondência com uma função em *JAVA* de mesmo nome, bem como estruturas de dados e outras implementações. Contudo, os primeiros testes realizados utilizando apenas a linguagem *JAVA* não se mostraram muito satisfatórios na implementação de algumas funções. Além disso, implementando os algoritmos para mapeamento a partir do código original em C/C++ permite ainda um ganho de performance, evitando que a aplicação *Android* tenha que realizar várias chamadas à *JNI* para transcrição de cada função *JAVA* implementando uma função C/C++ correspondente, diminuindo a sobrecarga (do inglês, *overhead*) de processamento da aplicação.

Assim, apenas um código fonte contendo todas as funções C/C++ de processamento de imagem utilizadas pela aplicação, requerendo poucas chamadas à *JNI*, apenas para troca eventual de dados entre o código *JAVA* e o código fonte em C/C++, se mostrou uma opção mais vantajosa. Devido à estas questões, optou-se pela programação baseada na dualidade *JAVA* e C/C++ como informado.

Maiores detalhes de utilização da biblioteca *OpenCV* ou das funcionalidades nativas fornecidas pelo sistema operacional *Android*, podem ser encontradas em OPENCV (2017b) e ANDROID (2017), respectivamente.

4.2 Funcionamento da aplicação

Após a integração do *OpenCV* com a aplicação *Android*, a biblioteca *OpenCV* pôde ser finalmente utilizada iniciando-se pela implementação da interface chamada *CvCameraViewListener2* pela atividade principal, *MainActivity*, da aplicação. Com a implementação desta interface, três métodos são fornecidos para utilização, e que devem ser sobrescritos, sendo eles: *onCameraViewStarted*, *onCameraViewStopped* e *onCameraFrame*. O método *onCameraViewStarted* permite as configurações iniciais da câmera com relação a dimensão da imagem capturada pela câmera e outros fatores. É geralmente neste método que se configura também uma estrutura de dados do tipo *Mat* (pertencente à biblioteca *OpenCV*) com base na dimensão da imagem a ser capturada e informada pelo próprio método. Esta estrutura do tipo *Mat* serve para acomodar cada imagem capturada, uma por vez, em uma representação que possa ser processada pela biblioteca *OpenCV*. Já o método *onCameraViewStopped* é utilizado para se decidir o que fazer quando a câmera for parada, como quando a aplicação é encerrada. No caso deste trabalho, o recurso da câmera é simplesmente liberado quando *onCameraViewStopped* for chamado.

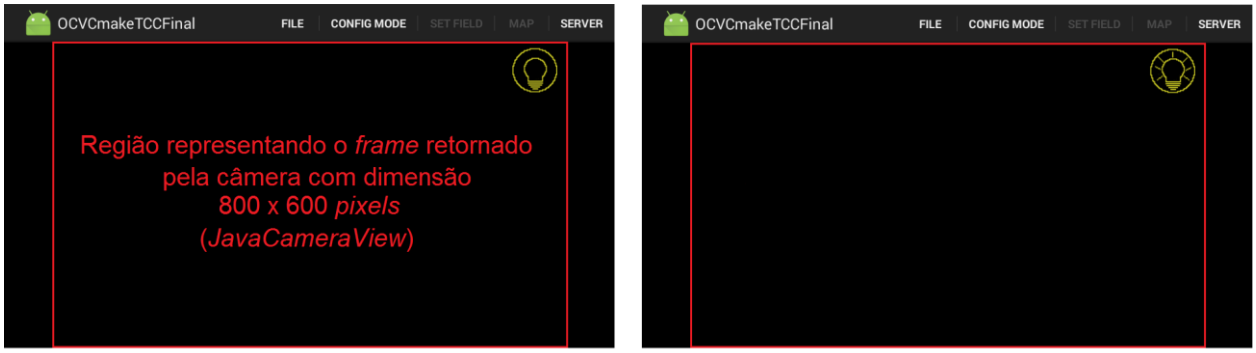
O terceiro método, *onCameraFrame*, é de particular interesse. É este método que é chamado pela interface *CvCameraViewListener2* implementada, quando uma imagem ou quadro (do inglês, *frame*) é capturada(o) pela câmera. Os termos imagem, quadro e *frame* são utilizados neste trabalho de forma equivalente.

O método *onCameraFrame* recebe todo *frame* capturado pela câmera e precisa retornar um outro *frame* ao final de sua chamada. O *frame* retornado é o que será visto pelo usuário. Para isso, no projeto da interface gráfica da aplicação, um elemento visual chamado *JavaCameraView* é inserido no *layout* principal da interface da aplicação. Assim, toda vez que *onCameraFrame* retornar um *frame*, este será apresentado ao usuário através do elemento *JavaCameraView*.

Como se pode observar, o método *onCameraFrame* fornece um ponto de entrada para todo o processamento de imagem a ser realizado com a imagem capturada, com objetivo final de se obter o mapa representante do campo na bancada de testes. É este método também que retorna a imagem processada pela aplicação e permite que o usuário a visualize e interprete seus resultados de forma visual. Assim, é no corpo do método *onCameraFrame* que todos os modos de operação da aplicação serão tratados. Estes modos de operações permitem que uma ou outra atividade seja realizada dentro do corpo do método *onCameraFrame*, e são definidos de acordo com os *menus* de configuração desenvolvidos para esta aplicação, presentes na interface principal da mesma. No total são seis modos de operação: *Crop and Goal*, *Obstacles*, *Grid*, *E-Puck*, *Original* e *Map*. Cada um deles será descrito a seguir conforme o *menu* de configuração a qual pertencem.

4.2.1 Interface principal da aplicação

A interface principal da aplicação é apresentada na Fig. 4.1, e é dividida em cinco *menus* principais, sendo eles: *FILE, CONFIG MODE, SET FIELD, MAP* e *SERVER*.



a) b)

Figura 4.1. Interface principal da aplicação, com região representativa do *frame* retornado pela câmera e posteriormente processado. a) com *flash* desligado; b) com *flash* ligado (O autor)

Toda interface foi projetada com utilização do idioma inglês, com intuito de facilitar sua disseminação em trabalhos futuros, além de ser o idioma padrão em desenvolvimento de *software*. Há além disso um botão ilustrado pela lâmpada em amarelo no canto superior direito, que pode ser pressionado a qualquer momento para ligar o *flash* da câmera (caso suportado pelo dispositivo) ou desligá-lo. O intuito do *flash* é aumentar as condições de iluminação em aplicações de baixa iluminação, podendo vir a ser útil em futuras utilizações. A Figura 4.1a ilustra a interface principal com o *flash* desligado, e a (b) com o *flash* ligado.

Observa-se ainda na Fig. 4.1a a região retangular em vermelho representando a *JavaCameraView*, ou seja, a região onde o *frame* processado e retornado pelo método *onCameraFrame* é apresentado ao usuário. É nesta região que o usuário poderá visualizar a imagem capturada do campo na bancada de testes e o processamento de imagem realizado sobre ela, bem como a tarefa de mapeamento em execução. É por ela também que o usuário terá um retorno visual de como as configurações realizadas no *menu CONFIG* afetam, em tempo próximo ao real, o processamento da imagem capturada. Neste caso é apresentada uma região de dimensões 800 x 600 *pixels* para o dispositivo utilizado. Esta região varia de dispositivo para dispositivo, ou mais especificamente, de câmera para câmera. Além disso, as imagens capturadas estão no formato *RGBA*, ou seja, estão representadas no espaço de cor *RGB* e possuem um canal extra, denominado *Alpha*, que é responsável por controlar a

transparência da imagem, porém, não é utilizado neste trabalho para as tarefas de processamento de imagem.

Cabe ressaltar que o retângulo em vermelho apresentado na Fig. 4.1 serve apenas para fins de explicação e é substituído pela própria imagem retornada por *onCameraFrame*. Assim, nos próximos tópicos, o retângulo não será apresentado nas figuras. Além disso, optou-se por não mostrar nenhuma imagem na região do *frame* por questões de praticidade.

4.2.2 Menu FILE

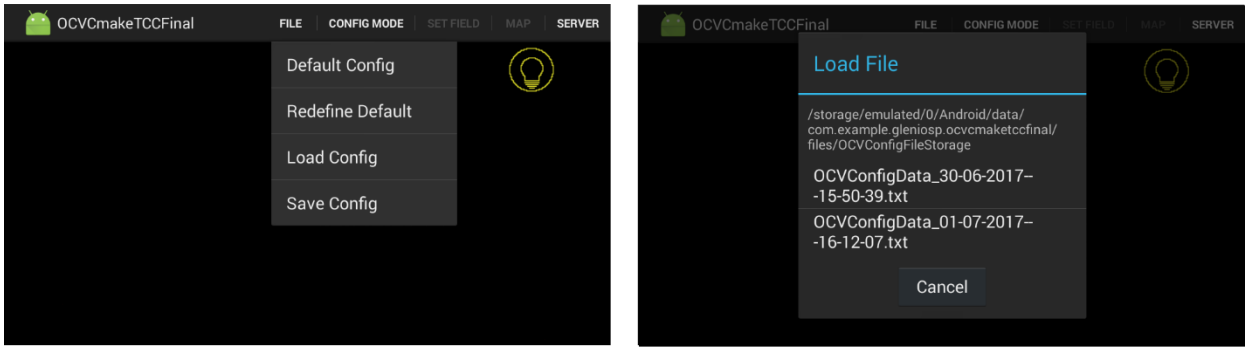
O *menu FILE* é responsável pelo gerenciamento de arquivos de configuração da aplicação. Todos os parâmetros de filtragem de cor do campo, do objetivo e dos obstáculos, bem como os parâmetros de busca pelo *E-Puck* e da quantidade de células utilizadas no mapeamento, além de todos os mapas gerados pela aplicação, podem ser salvos através do *menu FILE* ou utilizando-se funções empregadas pelo *menu FILE*. As opções fornecidas pelo *menu FILE* são apresentadas na Fig. 4.2.

O *menu FILE* contém quatro opções, como apresenta a Fig. 4.2a. A primeira opção, *Default Config*, permite carregar um arquivo de configuração padrão armazenado na memória interna do dispositivo. Estes arquivos de configuração são responsáveis pela configuração de todos os parâmetros de filtragem de cor, busca pelo *E-Puck* e decomposição em células do campo. Já a opção *Redefine Default* permite redefinir este arquivo padrão, salvando os dados dos parâmetros atuais em uso pela aplicação como o novo padrão.

A terceira opção, *Load Config*, utiliza-se de uma *Intenção* (recurso do sistema operacional *Android*) para abrir uma janela com os arquivos de configurações existentes na memória, como apresentado na Fig. 4.2b. É necessário apenas pressionar um destes arquivos de nome do tipo *OCVConfigData_(data)_(horário).txt* para que ele seja carregado pela aplicação e os parâmetros de configuração atuais assumam os valores presentes no arquivo escolhido. O botão *Cancel* fecha a janela.

Por fim, seguindo o mesmo padrão, a opção *Save Config* salva todos os parâmetros atuais em uso pela aplicação em um arquivo de nome do tipo *OCVConfigData_(data)_(horário).txt*, para futura utilização. Esta opção, também utiliza uma *Intenção* para abrir uma janela, permitindo escolher um outro nome para o arquivo caso desejado, além de fornecer um botão *Cancel* para fechar a janela, e um botão *Save* para de fato salvar o arquivo. O nome padrão *OCVConfigData_(data)_(horário).txt* dado aos arquivos

de configuração foi escolhido por ser intuitivo e único, contudo, podendo ser alterado, como já mencionado.



a) b)

Figura 4.2. *Menu FILE*. a) opções do *menu*; b) *Intenção* para carregamento de arquivo de configurações (O autor)

Cabe ressaltar que todos os arquivos são salvos na pasta de instalação do aplicativo na memória interna do dispositivo. Caso esse armazenamento não esteja disponível ou interditado por algum motivo, a aplicação fornece uma mensagem de aviso durante as operações de arquivo. Além disso, por estarem armazenados juntamente com os dados da aplicação, os arquivos salvos serão apagados caso a aplicação seja desinstalada.

4.2.3 Menu CONFIG

As configurações relativas à filtragem de cor, busca pelo *E-Puck* e divisão do campo em células é realizada no *menu CONFIG*. A Figura 4.3 apresenta este *menu*.

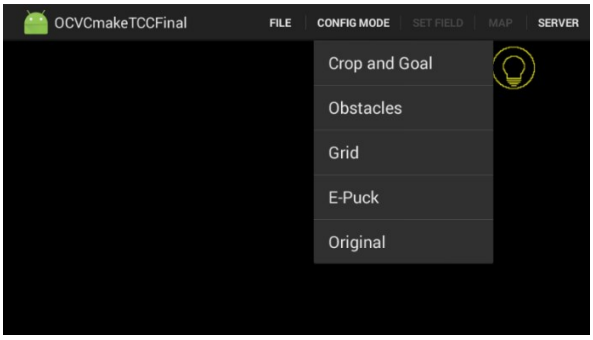


Figura 4.3. *Menu CONFIG* da aplicação (O autor)

O *menu CONFIG* contém cinco opções de configuração, ou melhor, cinco dos seis modos de operação possíveis em *onCameraFrame*. O modo *Original* é o único que não realiza processamento de imagem nenhum, servindo apenas para retornar o *frame* capturado pela câmera em sua forma original, sem nenhum processamento, para fins de visualização. Todos os demais modos de operação utilizam o recurso *fragmento* do sistema operacional *Android* para realizar as configurações em cada modo de forma transparente, sem que a Atividade principal perca o foco na tela. Assim, enquanto os modos são configurados, o processamento de imagem realizado por eles pode ser visto ao fundo através da *JavaCamaraView*, quase em tempo real.

4.2.3.1 Crop and Goal

É no *submenu* ou modo de operação *Crop and Goal* que o campo é filtrado a partir dos marcadores quadrados em vermelho, bem como o objetivo em vermelho. Esta filtragem é configurada por doze parâmetros, representando o valor mínimo e o valor máximo dos valores *H, S* e *V* do espaço de cor *HSV* utilizado para filtragem. No caso da cor vermelha, esta é representada por dois intervalos no espaço *HSV*. Os dois intervalos se devem ao fato de que no cone do espaço *HSV*, apresentado na Fig. 2.4, metade do intervalo da matiz vermelha se inicia em zero graus, e a outra metade se dá no final do intervalo de 360 graus da seção circular do cone, ou seja, a matiz vermelha é representada pelos dois extremos dos valores de matiz no cone do espaço *HSV*. Então, *onCameraFrame* recebe o *frame* capturado pela câmera e faz uma chamada à função nativa criada, *goalAndFieldExtraction*. Esta função inicialmente converte a imagem em *RGB* para *HSV* utilizando a função *cvtColor* da biblioteca *OpenCV*. Todos os *pixels* da imagem capturada são em seguida filtrados, deixando-se apenas aqueles que estiverem dentro dos limites de cada um dos dois intervalos *HSV* configurados no *fragmento* aberto por *Crop and Goal*, e acessados pela função *goalAndFieldExtraction*. Durante a filtragem dos *pixels*, operações morfológicas de fechamento são aplicadas à imagem para eliminação de pequenos contornos em vermelho esparsos, além de uma suavização da imagem via aplicação de um filtro gaussiano.

Com os marcadores e o objetivo em vermelho filtrados, formando uma imagem binarizada apenas com estes *pixels*, a função de identificação *findContours* da biblioteca *OpenCV* é empregada para identificação dos contornos dos marcadores e objetivo. Após identificação dos contornos, a função *approxPolyDP*, também implementada na biblioteca *OpenCV*, aproxima os contornos pelo menor polígono possível, e então estes polígonos são passados à *boundingRect*, disponível na biblioteca *OpenCV*, formando um retângulo para

cada marcador e para o objetivo. A localização e dimensão do retângulo representando o objetivo é salva em uma variável global dentro do código fonte contendo as funções nativas C/C++ criadas. E em seguida, uma região de interesse é traçada com vértices nos dois centros opostos mais distantes dos retângulos representando os marcadores, e então esta região é salva. É dentro da região de interesse que todos os outros quatro modos de operação operam: *Obstacles*, *Grid*, *E-Puck* e *Map*. A região de interesse e o objetivo são desenhados sobre a imagem capturada para visualização por parte do usuário.

Com o campo, ou região de interesse, encontrado, este pode ser definido pela aplicação e utilizado durante a execução dos outros modos de operação. Para definição do campo é necessário que o *menu SET FIELD* seja pressionado. Após pressionado o *menu* passará a indicar *UNSET FIELD*, e o campo se manterá fixo, podendo os marcadores serem retirados do campo, caso necessário. Para definir um novo campo para mapeamento, basta pressionar *UNSET FIELD*, que este volta a ser *SET FIELD*, permitindo que um novo campo seja definido novamente. O modo de operação *Crop and Goal* com o *fragmento* de configuração dos parâmetros *HSV* aberto, é apresentado na Fig. 4.4. É apresentado também na Fig. 4.4, o *menu* de definição de campo no estado *UNSET FIELD*, após o campo ter sido definido.

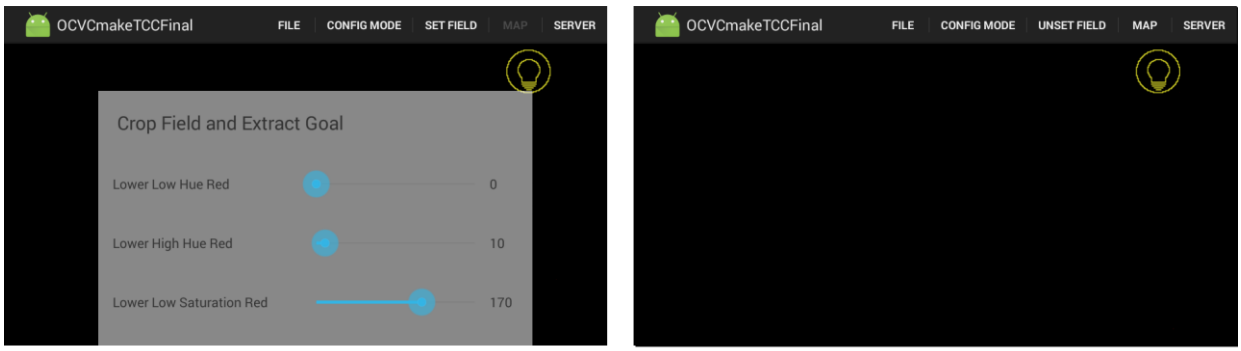


Figura 4.4. Modo de operação *Crop and Goal*. a) fragmento de configuração dos parâmetros *HSV*; b) *menu SET FIELD* no estado *UNSET FIELD* após definição do campo (O autor)

Os parâmetros de configuração no *fragmento* para o primeiro intervalo *HSV* com limite inferior vermelho são:

- *Lower Low Hue Red*: intervalo inferior do ponto inicial da matiz vermelha;
- *Lower High Hue Red*: intervalo inferior do ponto final da matiz vermelha;
- *Lower Low Saturation Red*: intervalo inferior do ponto inicial da saturação vermelha;

- *Lower High Saturation Red*: intervalo inferior do ponto final da saturação vermelha;
- *Lower Low Value Red*: intervalo inferior do ponto inicial da intensidade vermelha;
- *Lower High Value Red*: intervalo inferior do ponto final da intensidade vermelha.

Todos os parâmetros acima possuem ainda um correspondente que começa com *Upper* para o segundo intervalo *HSV* com limite superior vermelho. Além disso, no formato *HSV* os limites máximo e mínimo para cada componente na biblioteca *OpenCV* são:

- *Hue*: mínimo zero e máximo 179;
- *Saturation*: mínimo zero e máximo 255;
- *Value*: mínimo zero e máximo 255.

Assim, deve se ter esses valores mente durante a configuração, já que a implementação do espaço de cor *HSV* pela biblioteca *OpenCV* é diferente do usual teórico, que geralmente representa as componentes *H, S* e *V* com os seguintes máximos: 360, cem e cem, respectivamente.

4.2.3.2 Obstacles

Para identificação dos obstáculos, é necessário primeiro que o campo esteja definido após utilização do modo de operação *Crop and Goal*. Assim como o modo de operação *Crop and Goal*, o modo *Obstacles* fornece um *Fragmento* para realização da configuração dos parâmetros *HSV* para filtragem da cor preta. Contudo, o preto possui apenas um intervalo *HSV*, com valores mínimos e máximos de cada componente: *H, S* e *V*.

Durante o modo de operação *Obstacles*, a função nativa criada, *obstaclesExtraction*, é chamada, realizando inicialmente a filtragem da cor preta a partir dos parâmetros configurados no *fragmento*. Em seguida, a função *findContours* identifica os contornos dos objetos filtrados, e novamente é realizada uma simplificação por polígonos através de *approxPolyDP*. No próximo passo, é verificado se os polígonos produzidos são retângulos, ou seja, se possuem quatro vértices, e caso sim, suas localizações e dimensões são salvas, e os polígonos são desenhados sobre a imagem capturada para visualização.

Os parâmetros de configuração no *fragmento* para o intervalo *HSV* preto são:

- *Low Hue Black*: ponto inicial da matiz preta;

- *High Hue Black*: ponto final da matiz preta;
- *Low Saturation Black*: ponto inicial da saturação preta;
- *High Saturation Black*: ponto final da saturação preta;
- *Low Value Black*: ponto inicial da intensidade preta;
- *High Value Black*: ponto final da intensidade preta;

Cabe observar que se o campo não tiver sido definido, o *fragmento* aberto, apesar de configurável, não mostrará nenhum processamento sendo realizado e retornado para visualização na *JavaCameraView*. O processamento só se dará de fato, e só será visualizado, se o campo tiver sido definido. O mesmo acontece para os outros modos de operação: *Grid, E-Puck e Map*.

A Figura 4.5 apresenta o *fragmento* de configuração do modo *Obstacles*.

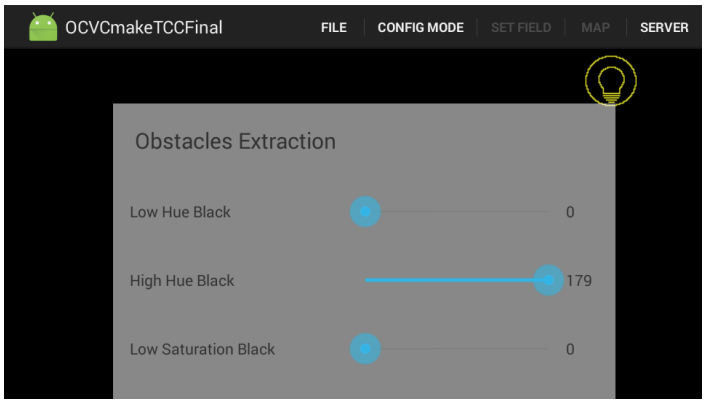


Figura 4.5. Modo de operação *Obstacles* (O autor)

4.2.3.3 Grid

O modo de operação *Grid* na verdade não emprega nenhuma técnica de processamento de imagem. Contudo, ele é parte fundamental para o mapeamento. Este modo também conta com um *fragmento*, sendo utilizado para configuração do número de células na direção Y (vertical) e na direção X (horizontal). O número mínimo de células é um, e o número máximo é cem. Desta forma, caso o campo esteja definido, a função *divideFieldInCells* é chamada e divide o campo em células retangulares de igual tamanho com base na quantidade de células em Y e X. Essa divisão nada mais é do que a construção de Y vezes X retângulos, onde estes são desenhados sobre a imagem para visualização pelo usuário, e suas

localizações e dimensões, salvas. Às células são atribuídas uma numeração de zero à Y vezes X menos um, começando do canto superior esquerdo e indo até o canto inferior direito. A Figura 4.6 apresenta o *fragmento* de configuração do modo *Grid*.

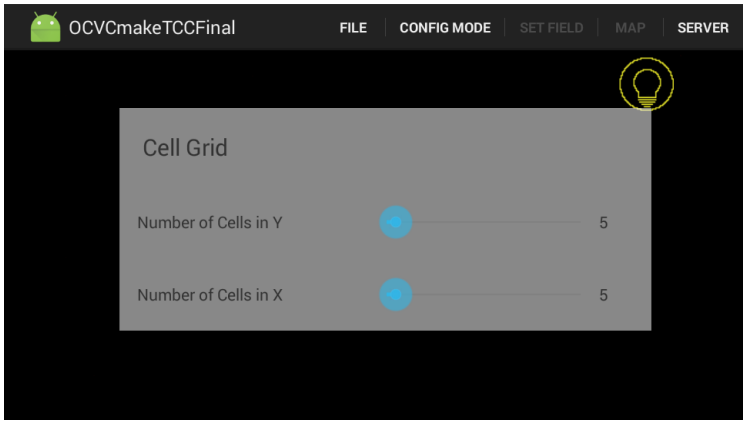


Figura 4.6. Modo de configuração *Grid* (O autor)

4.2.3.4 E-Puck

O modo de operação *E-Puck* realiza a identificação de robôs *E-Puck* em campo. Para isso, o campo precisa estar definido. Um *fragmento* é utilizado para configuração dos parâmetros necessários à identificação dos robôs *E-Puck*. A função *HoughCircles* da biblioteca *OpenCV* encontra os círculos presentes em uma imagem utilizando a Transformada circular de *Hough*, e devido ao formato circular de um robô *E-Puck*, este pode ser facilmente encontrado.

Assim, no modo de operação *E-Puck* é realizada a chamada à função nativa criada, *findEPuck*. Esta função primeiramente converte a imagem *RGB* para uma imagem em escala de cinza, utilizando a função *cvtColor*. Essa conversão é exigida pelo detector de bordas de *Canny* utilizado internamente na função *HoughCircles*. A imagem em escala de cinza é então repassada para uma outra função nativa criada, denominada *houghCircleTransform*. Os parâmetros informados no *fragmento* também são passados à ela. Esta função simplesmente aplica um filtro gaussiano a imagem em escala de cinza para suavização da imagem, de modo a reduzir os possíveis falsos círculos que possam ser retornados pela função *HoughCircles* na forma de ruídos. Então, após aplicação do filtro, *houghCircleTransform* chama, por fim, *HoughCircles* com a imagem em escala de cinza e os parâmetros de configuração que foram passados à *houghCircleTransform*. *HoughCircles* implementa internamente um detector de

bordas *Canny* para segmentação da imagem e prossegue com a identificação dos círculos a partir deste ponto.

A identificação dos círculos se dá com base nos parâmetros fornecidos à função *HoughCircles*. Este parâmetros, ajustados no *fragmento Android* aberto, são os próprios parâmetros requeridos pela implementação da função *HoughCircles* na biblioteca *OpenCV*, sendo estes:

- *Hough Circle minDist Divider*: distância mínima entre os círculos detectados. Neste caso, esta distância é usada como divisor na expressão: *número de linhas na imagem em escala de cinza* dividido por *Hough Circle minDist Divider*. O resultado desta expressão é passado ao parâmetro *minDist* da função *HoughCircles*. O valor mínimo de *Hough Circle minDist Divider* é um, e o máximo é cinquenta;
- *Hough Circle param1 (equivalente ao param1)*: valor específico utilizado pelo detector de bordas de *Canny* empregado na função *HoughCircles*. O mínimo é um e o máximo é 500;
- *Hough Circle param2 (equivalente ao param2)*: outro valor específico utilizado pelo detector de bordas de *Canny* empregado na função *HoughCircles*. O mínimo é um e o máximo é 500;
- *Hough Circle Minimum Radius (equivalente ao minRadius)*: raio mínimo dos círculos a serem identificados. Mínimo zero e máximo 200.
- *Hough Circle Maximum Radius (equivalente ao maxRadius)*: raio máximo dos círculos a serem identificados. Mínimo zero e máximo 200.

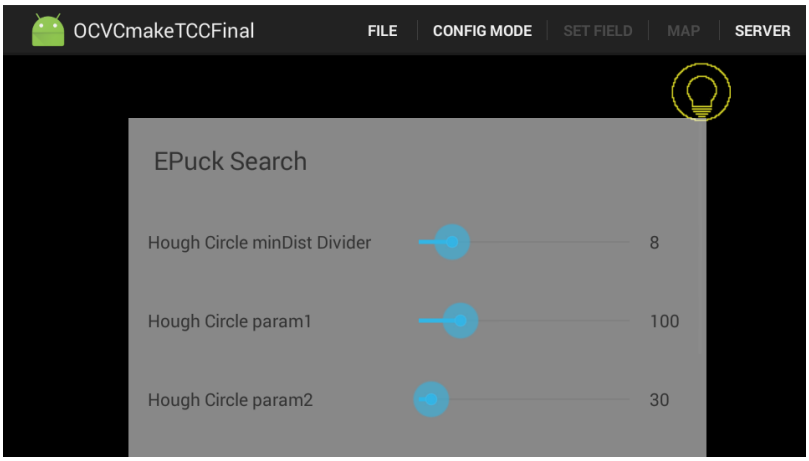


Figura 4.7. Modo de operação *E-Puck* (O autor)

HoughCircles identifica os possíveis círculos, que representam os *E-Pucks* na imagem, e retorna a localização do centro de cada um deles, bem como o raio destes centros. A identificação positiva dos robôs *E-Pucks* pode ser reforçada ao se restringir os raios mínimos e máximos a serem identificados, diminuindo as chances de que círculos na forma de ruídos possam ser identificados como um robô *E-Puck*. Por fim, com base nas informações dos círculos retornados por *HoughCircles*, um retângulo mínimo que envolva os círculos é calculado para cada um deles, e então a localização e dimensão destes retângulos é salva. Os retângulos construídos são também desenhados sobre a imagem para que o usuário da aplicação possa visualizá-los. Por fim, a Figura 4.7 apresenta o *fragmento* de configuração *E-Puck*.

4.2.4 Menu MAP

O mapeamento ocorre de fato com as configurações realizadas neste *menu*, que por sua vez, é o último dos seis modos de operação possíveis da aplicação. Após configurar todos os parâmetros necessários nos outros *submenus* ou modos de operação apresentados, o mapeamento pode ser iniciado. Este *menu* só estará disponível caso o campo tenha sido definido anteriormente. A partir deste ponto, pode-se selecionar a opção *Start Mapping* no *menu MAP* e o mapeamento se iniciará.

Ao se selecionar *Start Mapping*, todas as funções dos outros modos de operação são chamadas com os parâmetros de configuração atuais de todos eles. Assim, nesta ordem, as funções: *obstaclesExtraction*, *findEPuck* e *divideFieldInCells* são chamadas. Vale ressaltar que o campo já foi previamente definido, e o objetivo, encontrado e salvo. Após chamada à todas estas funções, todos os objetos que foram encontrados, serão mapeados com uma chamada à função nativa criada, *mapField*. Esta função varre todos os objetos identificados, representados por retângulos, e procura por todas as intersecções destes com os retângulos das células criadas. O mapeamento se dá então por meio da maior área de intersecção encontrada, ou seja, a célula que possuir a maior área de intersecção com determinado objeto, será definida com o estado representativo deste objeto. Assim, se a maior área de intersecção da célula de número vinte é com um robô *E-Puck* no campo, por exemplo, é atribuído o estado *epuck* à esta célula. Os outros estados possíveis são: *obstacle* e *goal*. Todas as demais células são consideradas livres. Essa informação de mapeamento é salva no formato *estado:número_da_célula* e é retornada para o código em *JAVA*, dentro do método *onCameraFrame*. Além disso, todas as intersecções dos objetos com as células são destacadas desenhando-se um retângulo verde para cada intersecção encontrada do objetivo

e dos robôs *E-Puck* com as células, e um retângulo azul para cada intersecção encontrada dos obstáculos com as células. A *JavaCameraView* apresenta o resultado do mapeamento ao usuário. Todos estes passos são repetidos para cada *frame* recebido da câmera, fornecendo um mapeamento contínuo, que considera também qualquer deslocamento de posição que os robôs *E-Puck* possam sofrer.

O *menu MAP* é apresentado na Fig. 4.8. Observa-se nesta figura que é possível selecionar apenas *Start Mapping* inicialmente. Após se selecionar *Start Mapping*, outras opções são habilitadas, mais especificamente, as opções de configuração do *Timer*. Além disso, *Start Mapping* se torna *Stop Mapping*, para que o mapeamento possa ser parado a qualquer momento, ou caso contrário, este executará indefinidamente. O *Timer* (ou temporizador) é utilizado nesta aplicação para que seja possível salvar o mapa gerado e/ou enviá-lo via protocolo de comunicação *Bluetooth* para qualquer aplicação conectada à esta aplicação, através de intervalos de tempo regulares e durante um determinado tempo estabelecido pelo usuário.

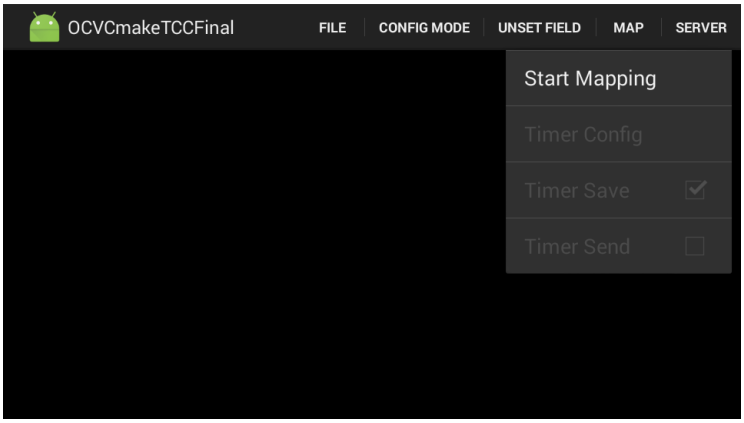


Figura 4.8. *Menu MAP* relativo ao modo de operação *Map* (O autor)

A opção *Timer Config* do *menu* utiliza o recurso de *Intenção* do SO *Android*, assim como foi utilizado no *menu FILE*. A *Intenção* gerada permite que se escolha a quantidade máxima de tempo em minutos e segundos em que a aplicação estará salvando em sua memória interna os dados de mapeamento e/ou enviando esses dados via *Bluetooth* à clientes conectados à ela. Dentro deste tempo estabelecido o usuário pode escolher o intervalo de tempo em que essas ações serão realizadas até que o tempo limite seja atingido. A *Intenção* gerada é apresentada na Fig. 4.9d.

Observa-se na Figura 4.9 que as configurações de envio e salvamento dos mapas gerados são realizadas por duas caixas de seleção. É possível escolher apenas salvar os mapas, apenas enviar os mapas, e salvar e enviar os mapas. Pelo menos uma das opções deve estar selecionada, e caso o usuário desmarque todas, a aplicação por padrão, marca a caixa de seleção de salvar os mapas. Esses mapas são salvos na memória interna do dispositivo, na pasta de instalação da aplicação, da mesma maneira que os arquivos de configuração, porém em pastas separadas. A pasta dos mapas possui o nome *OCVMapFileStorage*, enquanto a dos arquivos de configuração possui o nome *OCVConfigFileStorage*. Quando a opção de salvar mapas está marcada, para cada mapeamento produzido a partir de um *frame* capturado pela câmera, são salvos um arquivo de imagem contendo o mapa gerado e um arquivo de texto com a informação *estado:número_da_célula* em linhas individuais, durante cada intervalo de tempo configurado no *timer*. O nome destes arquivos é único e é dado, para a imagem e para o arquivo de texto, respectivamente, por: *OCVMap_(data)_(horário).png* e *OCVMap_(data)_(horário).txt*. No caso do envio via *Bluetooth*, apenas o arquivo de texto é enviado.

A Figura 4.9 de (a) à (c) apresenta a seleção das opções de salvar ou enviar mapa.

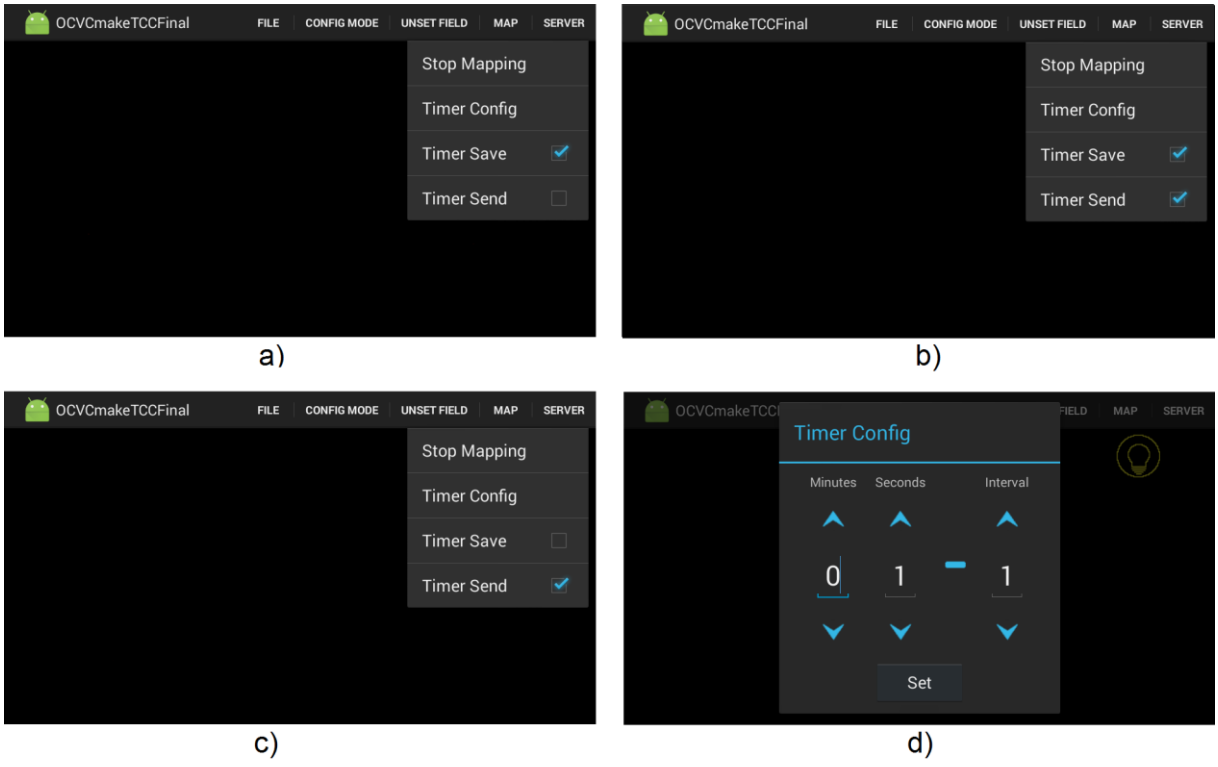


Figura 4.9. *Menu MAP* com todas as opções habilitadas. a) mapeamento em execução com apenas a opção de salvar mapa marcada; b) mapeamento em execução com as opções de

salvar mapa e enviar mapa marcadas; c) mapeamento em execução com apenas a opção de enviar mapa marcada; d) *Intenção* de configuração do *timer* com o primeiro algarismo relativo aos minutos, o segundo aos segundos, e o terceiro ao intervalo de tempo (O autor)

4.2.5 Menu SERVER

O *menu SERVER* é o responsável por todas as funcionalidades de comunicação *Bluetooth* da aplicação. A aplicação atua como um servidor *Bluetooth*, permitindo que qualquer dispositivo *Bluetooth* possa se conectar à ela e receber as informações de mapeamento produzidas, sendo sete o número máximo de clientes que podem se conectar ao servidor. Para que um dispositivo possa se conectar à esta aplicação, é necessário que esta primeiramente habilite o módulo *Bluetooth* através da opção *Enable Bluetooth*, conforme apresentado na Fig. 4.10. Após o módulo *Bluetooth* ser habilitado, as demais opções do *menu SERVER* ficam disponíveis. A opção *Enable Bluetooth* também se torna *Disable Bluetooth,* para que o módulo possa ser desabilitado posteriormente.

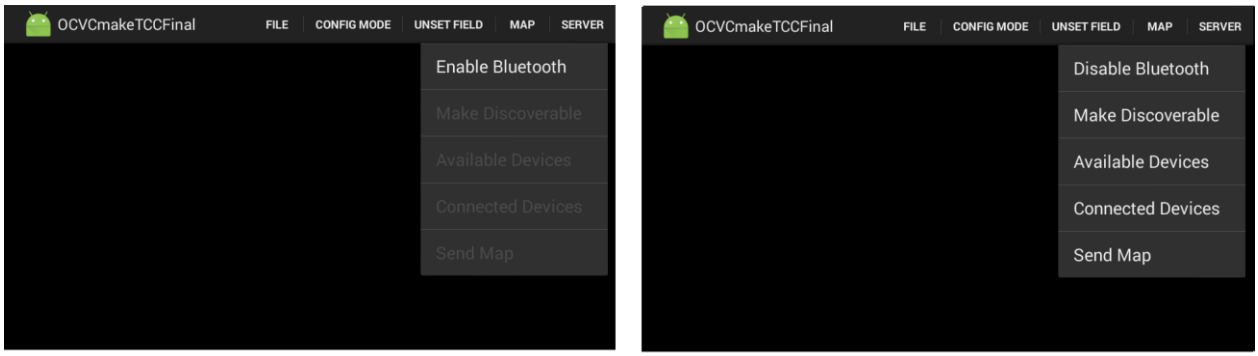


Figura 4.10. *Menu SERVER*. a) menu com o Bluetooth desabilitado; b) menu após habilitar o *Bluetooth* (O autor)

Assim que o *Bluetooth* é habilitado, a aplicação inicia um servidor *Bluetooth* apto a receber conexões de clientes remotos. Para que um cliente possa se conectar ao servidor da aplicação, é necessário que este informe o endereço *MAC* do adaptador *Bluetooth* do servidor, o *UUID* do tipo *Serial Port Profile (SPP)* e o canal *RFCOMM* em que o servidor estará escutando por conexões. Essas informações podem ser obtidas por um cliente remoto ao executar um serviço de descobrimento *Bluetooth*. Para isso, o servidor *Bluetooth* precisa estar visível ou descoberto. Essa ação pode ser realizada selecionando-se a opção *Make*

Discoverable no *menu SERVER*. O servidor *Bluetooth* ficará descoberto por cinco minutos. Após ter sido encontrado por um algum dispositivo *Bluetooth* remoto, um possível cliente, este dispositivo pode tentar se conectar ao servidor. O servidor irá então enviar uma solicitação de emparelhamento para este dispositivo cliente, e caso este confirme a troca de chaves, cliente e servidor serão emparelhados, e caso não, a tentativa de conexão é descartada. Cabe ressaltar que para um cliente que já esteja emparelhado com o servidor *Bluetooth*, não é necessário que um serviço de descobrimento seja executado, e nem que o servidor esteja visível, bastando apenas que o cliente informe os dados corretos de conexão com o servidor.

Com o processo de conexão concluído, o servidor pode enviar dados de mapeamento para o cliente conectado, e também ler dados enviados pelo cliente. O envio de dados de mapeamento por parte do servidor pode ser realizado via *menu MAP* com a opção *Timer Send* marcada ou através da opção *Send Map* presente no *menu SERVER*, conforme informado adiante.

O *menu SERVER* conta ainda com outras três opções: *Available Devices*, *Connected Devices* e *Send Map*. A opção, ou *submenu*, Available *Devices*, abre uma *Intenção Android* que retorna todos os dispositivos *Bluetooth* emparelhados com o servidor em uma lista e permite varrer o ambiente, dentro das limitações de alcance do adaptador *Bluetooth*, em busca de possíveis clientes remotos que estejam visíveis. A varredura é realizada através do botão *Scan devices*, como apresentado na Fig. 4.11.

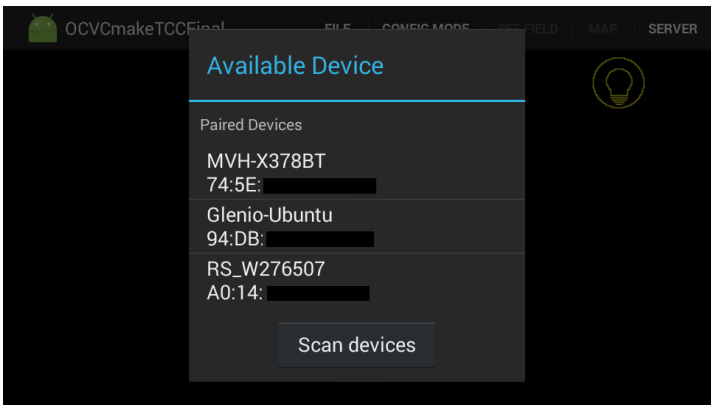


Figura 4.11. *Intenção Android* iniciada para verificação dos dispositivos *Bluetooth* emparelhados com o servidor e de dispositivos *Bluetooth* visíveis. Os endereços *MAC* foram apagados parcialmente por questões de segurança (O autor)

Na opção *Connected Devices* uma *Intenção Android* é aberta, como a em *Available Devices*, porém, para verificação dos dispositivos conectados ao servidor, em forma de lista.

É possível que estes dispositivos sejam desconectados do servidor ao se selecionar qualquer um deles na lista e confirmar a desconexão.

Já a opção *Send Map* permite selecionar um arquivo de mapeamento na memória interna do dispositivo *Android* e enviá-lo à todos os dispositivos clientes conectados ao servidor. Útil para testes, mesmo que a aplicação não esteja realizando mapeamento nenhum.

Caso o *Bluetooth* seja desabilitado, todos os dispositivos conectados ao servidor são desconectados automaticamente. Por fim, um fluxograma ilustrando o processo de comunicação *Bluetooth* do ponto de vista do servidor é apresentado na Fig. 4.12a, e do ponto de vista do cliente, na Fig. 4.12b.

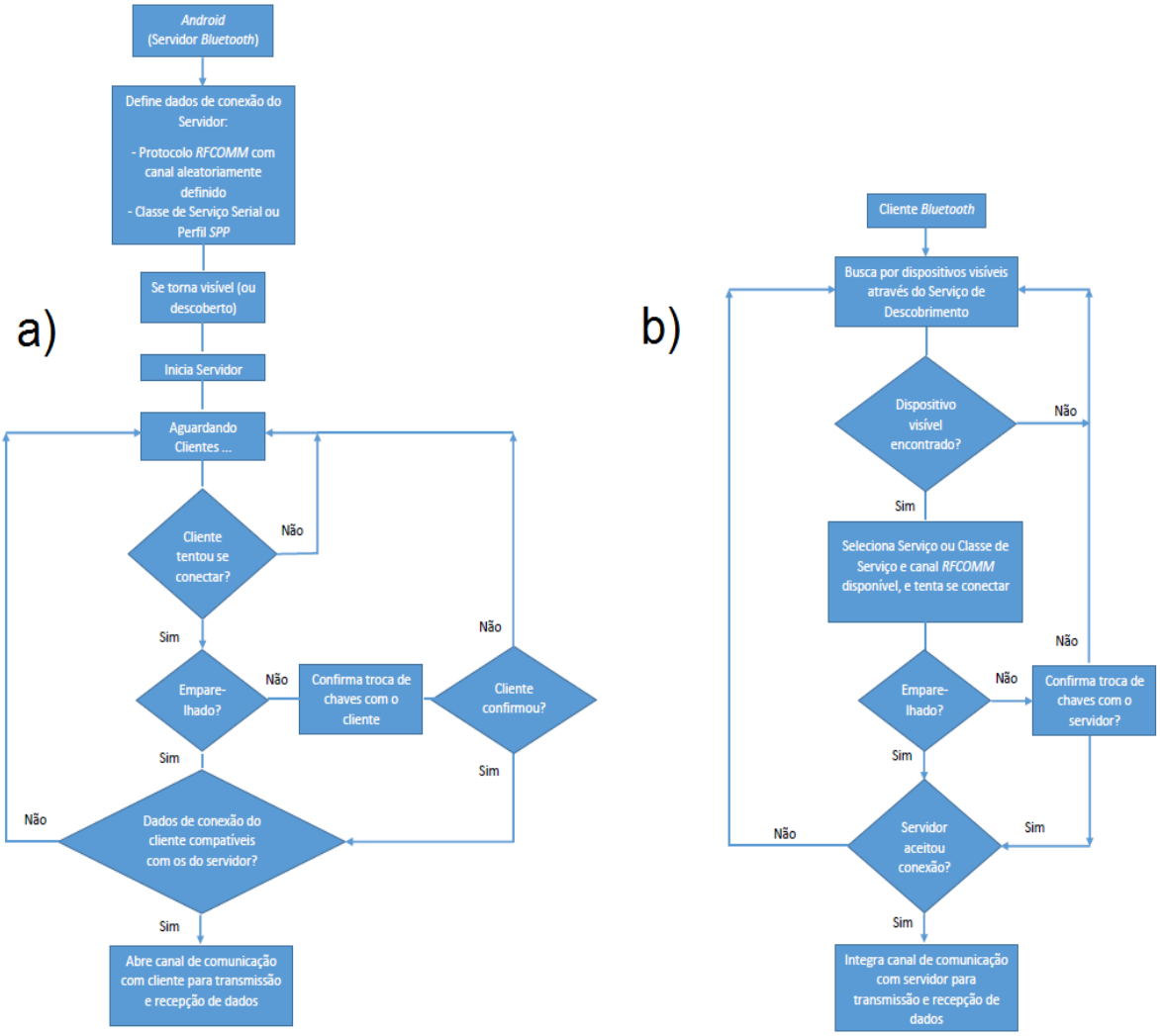


Figura 4.12. Comunicação cliente-servidor *Bluetooth*. a) fluxograma do processo de comunicação *Bluetooth* realizado pelo servidor; b) fluxograma do processo de comunicação *Bluetooth* realizado por um cliente remoto (O autor)

4.3 Considerações finais

Como delineado neste capítulo, a aplicação funciona a partir de um paradigma de programação baseado em evento. Ao se abrir a aplicação, esta inicia a câmera, realiza procedimentos de configurações iniciais e começa a captura das imagens. O ponto central do processamento realizado pela aplicação é a captura das imagens pela câmera, com chamada subsequente ao método *onCameraFrame*. Este método, ao receber a imagem capturada atribui-a à estrutura de dados *Mat* da biblioteca *OpenCV*, criada em *onCameraViewStarted*, de modo a ser processada pelos algoritmos de mapeamento desenvolvidos nesta aplicação. Conforme o modo de operação ou configuração escolhido na interface da aplicação, uma tarefa diferente é realizada pelo código de execução presente no corpo do método *onCameraFrame*.

Deste modo, todo *frame* recebido pela câmera do dispositivo é capturado e apresentado na *JavaCameraView*, após ser processado. Durante este processo, a aplicação pode ser configurada a qualquer momento através dos *menus* de sua interface, afetando diretamente o processamento das imagens capturadas. Todas estas configurações podem ainda serem salvas e carregadas a qualquer momento. Além disso, a captura de imagens da câmera é contínua, se assemelhando a gravação de um vídeo enquanto a aplicação estiver em funcionamento. Assim, um fluxo contínuo de chamadas a *onCameraFrame* é realizado, permitindo também um fluxo contínuo de mapas gerados. Com os mapas gerados é possível salvá-los na memória interna do dispositivo *Android* e/ou enviá-los a outros dispositivos via comunicação *Bluetooth*.

É de se observar que a aplicação possui uma enorme flexibilidade de configuração. Além disso, a aplicação é bem modular, permitindo que diversas modificações sejam realizadas sem afetar o funcionamento da aplicação como um todo. Uma das possíveis modificações em trabalhos futuros poderia ser a alteração da forma com que os mapas são estruturados, atualmente como, *estado:número_da_célula*, permitindo alguma outra estruturação desejada conforme projeto de pesquisa. Para que uma nova forma de estrutura de mapeamento seja definida, é necessário apenas reescrever parte do código na função nativa criada, *mapField*. Há ainda algumas melhorias que podem ser realizadas com relação a usabilidade da aplicação, em especial com relação ao modo de configuração dos parâmetros no *menu CONFIG*. A configuração dos parâmetros é realizada por barras deslizantes, e apesar da praticidade, um ajuste fino não é tão simples. Isso pode ser resolvido,

por exemplo, com a implementação de botões laterais do tipo seta, para incremento e decremento dos valores dos parâmetros de um em um.

Como conclusão parcial, um fluxograma resumido de todo o fluxo de processamento realizado pela aplicação, no que tange ao processamento das imagens capturadas, é esquematizado através da Fig. 4.13. Observa-se que para cada modo de operação, *Crop and Goal*, *Obstacles*, *Grid*, *E-Puck*, *Original* ou *Map*, um conjunto de tarefas diferente é executado.

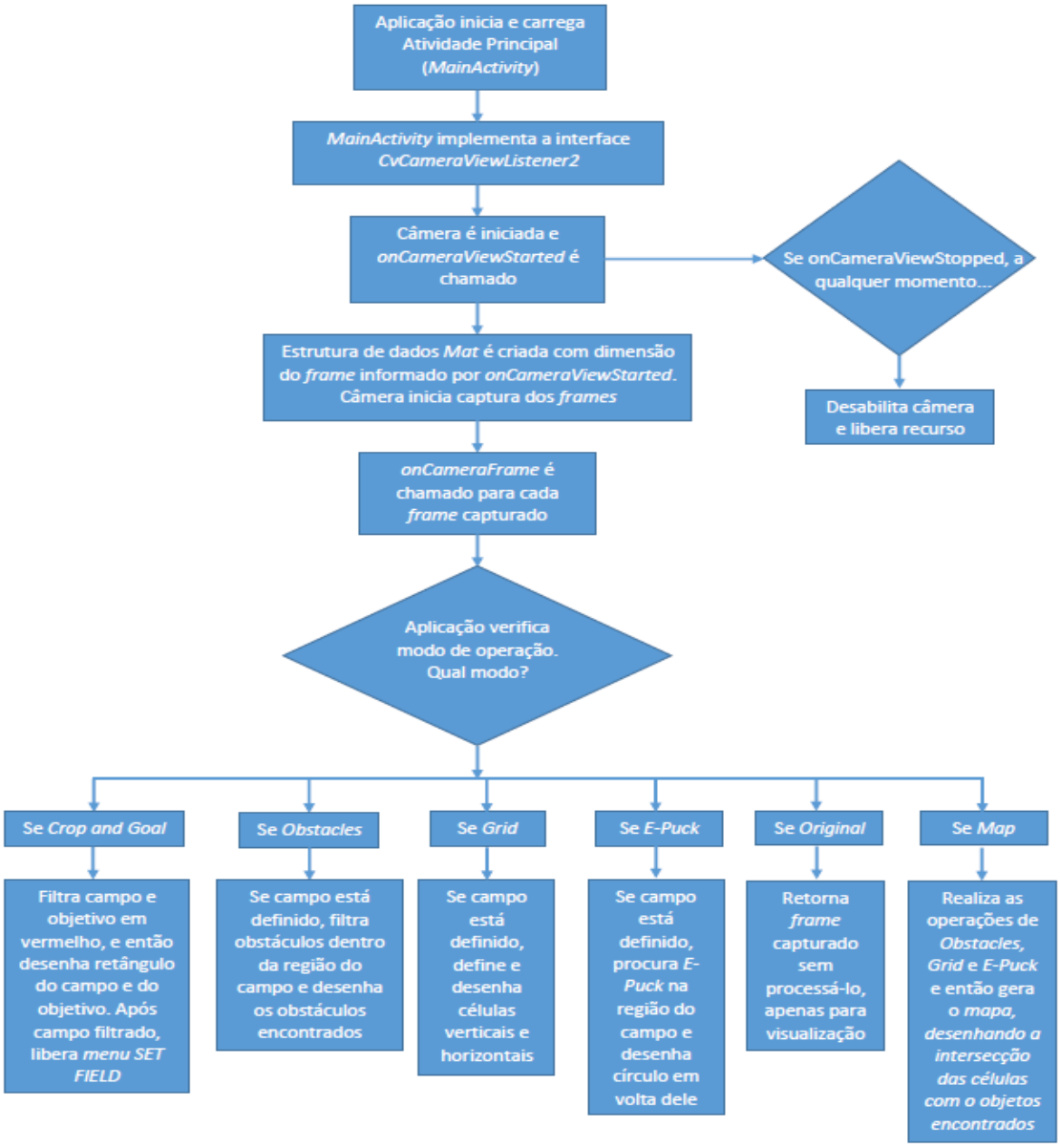


Figura 4.13. Fluxograma resumido dos modos de operação empregados pela aplicação após a captura das imagens pela câmera do dispositivo *Android* (O autor)

CAPÍTULO V

METODOLOGIA

Os métodos e materiais utilizados para estudo e desenvolvimento da aplicação desenvolvida são apresentados neste capítulo. São exemplificadas também as condições e configurações utilizadas na coleta dos resultados para posterior interpretação e validação da aplicação *Android*.

5.1 Bancada ou ambiente de testes

Para realização dos experimentos, uma bancada de testes foi utilizada. Esta bancada foi construída pelos integrantes do Laboratório de Computação Bio-Inspirada da Faculdade de Computação (FACOM) da Universidade Federal de Uberlândia (UFU), e é apresentada na Fig. 5.1. Já a Figura 5.2 apresenta um desenho esquemático com as dimensões relevantes da bancada. Conforme se observa nas Figura 5.1 e Figura 5.2, uma bancada simples foi utilizada, onde um campo de madeira *MDF (Medium Density Fiberboard)* foi colocado sobre ela. Na lateral da bancada um tubo de *PVC (Policloreto de vinila)* com uma junção em L constituem o suporte para inserção do *smartphone* ou dispositivo *Android*. No tubo de *PVC* próximo ao rasgo de inserção do dispositivo *Android*, há ainda um furo para passagem de um cabo *USB* que pode ser conectado ao dispositivo. Essa abordagem do cabo foi utilizada para se poder controlar a aplicação *Android* executando no dispositivo através de um computador pessoal com conexão *USB*. Contudo, a abordagem do cabo é opcional, já que existem aplicativos *Android* para controle remoto do *smartphone* via conexão sem fio do tipo *WiFi*.

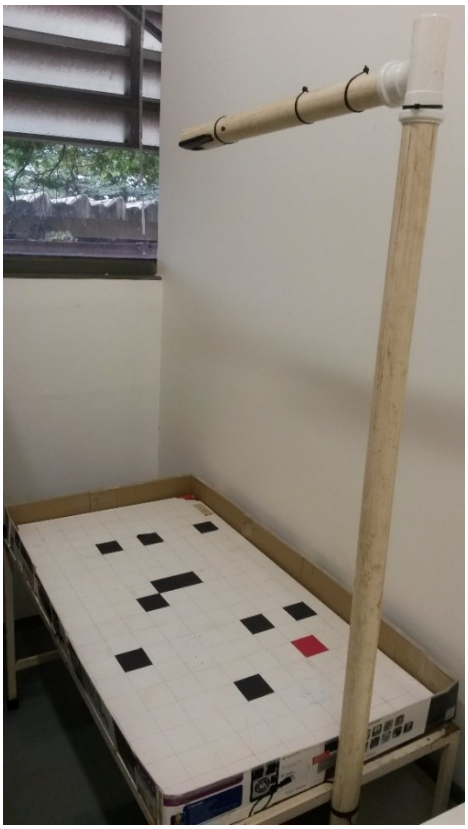


Figura 5.1. Bancada de testes construída pelos integrantes do Laboratório de Computação Bio-Inspirada (O autor)

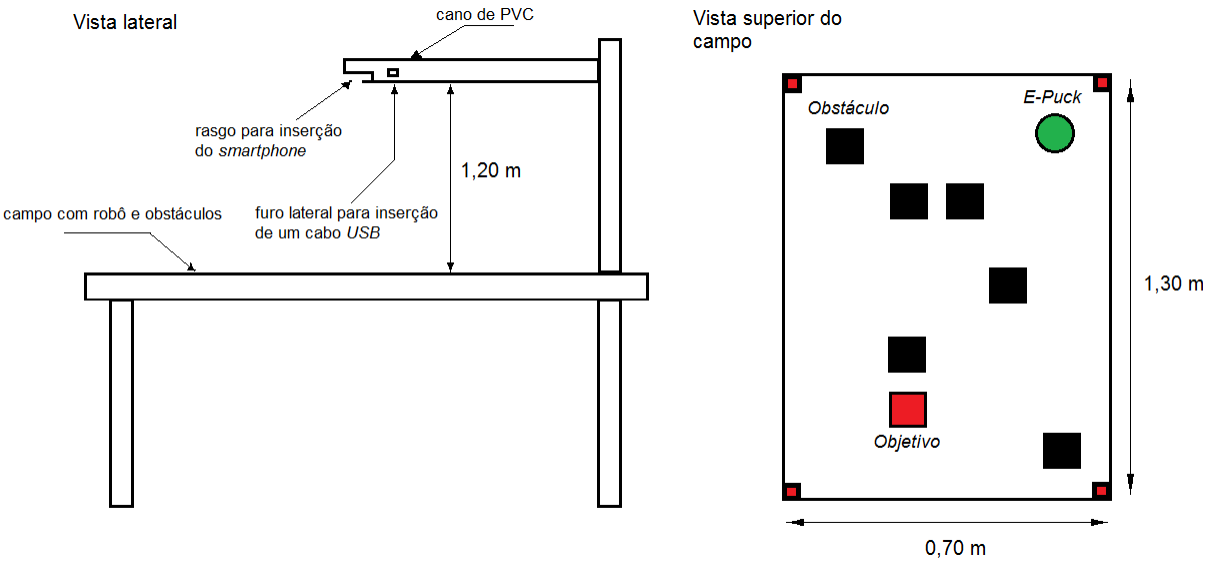


Figura 5.2. Desenho esquemático com as dimensões relevantes da bancada de testes (O autor)

No campo foram posicionados marcadores quadrados vermelhos para identificação das bordas do campo, já que a imagem capturada pela câmera engloba mais do que apenas o campo, sendo necessária uma filtragem para delimitação do campo de testes. Os obstáculos eram quadrados pretos, e o objetivo a ser atingido pelo robô foi representado por um quadrado em vermelho. O *E-Puck* é um robô de formato circular e foi representado na Fig. 5.2 como um círculo verde com a mesma dimensão de um obstáculo.

5.2 Configuração da bancada de testes e da aplicação *Android*

Para realização dos testes e obtenção dos resultados, o *smartphone* foi inserido no rasgo do tubo *PVC* com sua câmera traseira voltada para a bancada, ou mesa. Um cabo *USB* foi conectado ao *smartphone* e então passado pelo furo no tubo *PVC*, próprio para este cabo, conectando-se na outra ponta à um *notebook*. No *notebook* foi utilizado um aplicativo instalado no navegador *web Chrome*, chamado *Vysor*. Com o *Vysor* é possível controlar remotamente o *smartphone* e iniciar a aplicação *Android* para realização das devidas configurações durante a captura das imagens. No campo, foram colocados os marcadores de bordas em vermelho, enquanto os obstáculos e objetivo foram posicionados aleatoriamente no campo. A partir deste ponto, o robô *E-Puck* pôde ser colocado no campo e o processo de captura de imagens e mapeamento foi iniciado. No caso dos testes realizados, o robô *E-Puck* foi deslocado manualmente durante a captura das imagens, de modo a se verificar a capacidade da aplicação em identificá-lo, mesmo que este se desloque. Uma opção seria ainda programar o *E-Puck* para percorrer o campo de forma aleatória durante a captura das imagens.

Com as configurações iniciais realizadas, o processo de mapeamento se deu através dos seguintes passos, conforme já explicado detalhadamente no capítulo IV:

- Os marcadores do campo foram identificados através da filtragem da cor vermelha e da identificação de seus contornos via *findContours*;
- Com a identificação dos marcadores, o objetivo em vermelho foi identificado também, e seu contorno aproximado por um retângulo, sendo sua posição, salva;
- Definiu-se o campo virtual formado pelo retângulo com vértices nos centros dos dois marcadores opostos mais distantes no campo real;

- Após identificação do campo virtual, os obstáculos em preto foram filtrados dentro da região de interesse, o próprio campo virtual, com seus contornos individuais aproximados por retângulos e suas posições salvas em seguida;
- Em seguida, a localização do robô *E-Puck* foi encontrada através da Transformada circular de *Hough*, seu formato circular foi aproximado por um retângulo mínimo que o envolvesse, e então, sua posição foi salva;
- Com as posições do objetivo, dos obstáculos e do *E-Puck*, o campo foi decomposto em células de tamanhos iguais. A quantidade de células nas direções horizontal e vertical pode ser configurada a qualquer momento na aplicação;
- Com o campo subdividido em células, um algoritmo varreu todos os objetos encontrados, representados por retângulos, e verificou a quais células estes objetos pertenciam. Para cada célula contendo um dos objetos, foi atribuído um dos seguintes estados à célula: *epuck*, *obstacle* (*obstáculo*) ou *goal* (*objetivo*). Todas as células que não possuíssem um destes estados, foram consideradas células livres;
- Após definição dos estados das células, um mapa foi gerado com estas informações, sendo este salvo na memória interna do dispositivo *Android,* e podendo também ser enviado ao robô (ou qualquer outro dispositivo conectado à aplicação) via *Bluetooth*;
- Com o mapa salvo, todos os passos acima foram repetidos para cada imagem capturada pela câmera, a partir da definição do campo virtual, que se manteve estático, mesmo que os mercadores fossem removidos, mantendo-se assim a região de interesse.

Cabe ressaltar que a qualquer momento é possível reiniciar o campo virtual e defini-lo novamente, além de também ser possível alterar-se as configurações de filtragem de cor, quantidade de células e parâmetros da Transformada circular de *Hough*. Acrescenta-se também o fato de todos os objetos terem sido aproximados por retângulos para uma melhor identificação do seu posicionamento em relação às células do campo, já que estas também possuem formato retangular. Reafirma-se, por fim, que durante todos passos executados acima pela aplicação, o robô *E-Puck* pôde ser deslocado de sua posição, seja manualmente ou através de algum algoritmo executado por ele, permitindo o teste de identificação dinâmica do robô.

CAPÍTULO VI

RESULTADOS E DISCUSSÃO

Este capítulo apresenta os resultados obtidos empregando-se a metodologia informada no capítulo V. Ao final do capítulo os resultados são interpretados e discutidos.

6.1 Resultados obtidos

Para execução da aplicação e realização dos testes, um aparelho *smartphone* modelo *Samsung Galaxy S4 mini duos (GT-I9192)*, apresentado na Fig. 6.1, foi utilizado. Este *smartphone* possui as seguintes especificações técnicas:

- Sistema operacional: *Android KitKat* 4.4.2 (atualizado)
- Dimensões: 124,6 x 61,3 x 8,9 mm
- Massa: 107 g
- Dimensão da tela: 4.3 pol
- Resolução: 540 x 960 *pixels*
- *Chipset*: *Qualcomm MSM8930AB Snapdragon 400*
- Processador (*CPU*): *dual-core* 1,7 GHz *Krait* 300
- *GPU*: *Adreno* 305
- Memória interna: 8 GB
- Memória *RAM*: 1,5 GB

- Câmera primária (traseira): 8 *Megapixels* com autofoco, f/2.6, *LED (Light Emitting Diode) flash* e sensor *CMOS*
- Câmera secundária (frontal): 1.9 MP
- *WLAN (Wireless Local Area Network)*: *Wi-Fi* 802.11 a/b/g/n, *dual-band*
- *Bluetooth*: 4.0
- *USB*: *microUSB* 2.0
- Sensores: acelerômetro, giroscópio, proximidade e bússola
- Bateria: *Li-Ion* 1900 mAh removível

Figura 6.1. *Smartphone Galaxy S4 mini* (GSMARENA, 2013)

Apesar da resolução da câmera, as imagens capturadas pela aplicação *Android* utilizando a câmera traseira deste dispositivo possuem resolução de 800 x 600 *pixels*.

O *smartphone S4 mini* foi inserido no rasgo correspondente do suporte da mesa de testes, e um cabo *microUSB/USB-A* de três metros, com filtro, foi conectado à ele e a um *notebook*. O aplicativo *Vysor* instalado no navegador *Chrome* no *notebook* foi utilizado para controle da aplicação *OCVCmakeTCCFinal*, instalada no *smartphone*. Em seguida, foram dispostos os seguintes objetos no campo sobre a mesa: um robô *E-Puck*, nove obstáculos e um objetivo; além dos quatro marcadores quadrados vermelhos nos cantos do campo. Um

dos obstáculos utilizados possuía a dimensão de dois obstáculos. A configuração do campo é visualizada na Fig. 6.2.

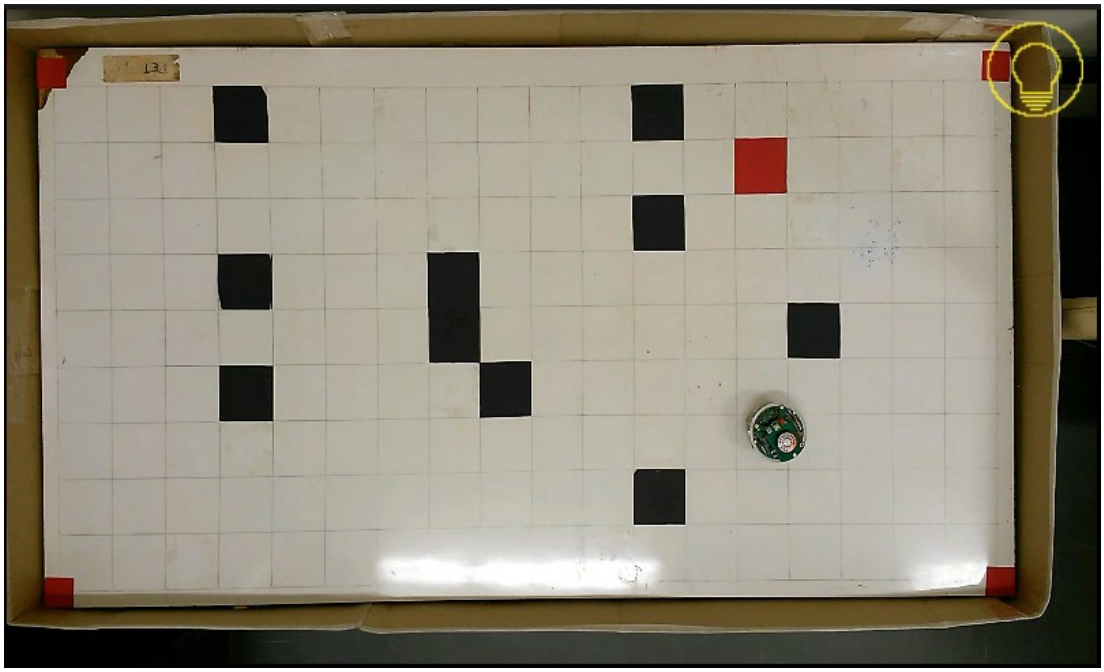


Figura 6.2. Configuração do campo utilizado nos testes (O autor)

Com os objetos dispostos em campo, a aplicação desenvolvida foi iniciada através do próprio aplicativo *Vysor* instalado no *notebook*. Os adaptadores *Bluetooth* no *smartphone* e no *notebook* foram habilitados e estes emparelhados, e também conectados via servidor *Bluetooth* iniciado pela aplicação desenvolvida. Assim, inicialmente o modo de operação *Crop and Goal* foi utilizado e os parâmetros *HSV* ajustados até que os quatro marcadores e o objetivo fossem encontrados, retornando-se também a região de interesse formada por estes marcadores. Encontrando-se a região de interesse interior aos marcadores, o *menu SET FIELD* da aplicação foi utilizado para definição do campo filtrado.

Seguindo-se a definição do campo, o modo de operação *Obstacles* do *menu CONFIG* foi utilizado para identificação dos obstáculos, variando-se os parâmetros *HSV* do *fragmento* aberto por ele. O próximo passo foi a identificação do robô *E-Puck* através do modo de operação *E-Puck*, variando-se os parâmetros fornecidos até que o *E-Puck* fosse encontrado. Para identificação do *E-Puck* as configurações padrões pré-inseridas na instalação da aplicação foram suficientes, sendo necessário configurar apenas os parâmetros de raio mínimo e máximo da Transformada circular de *Hough*. Com todos os parâmetros de

identificação dos objetos configurados, de forma a se obter o maior número de objetos identificados, a quantidade de células a ser usada na divisão do campo foi configurada através do modo de operação *Grid*.

Para geração dos mapas e coleta dos resultados, três configurações de *Grid* foram utilizadas, dividindo o campo em:

- Vinte células: quatro na vertical e cinco na horizontal;
- 49 células: sete na vertical e sete na vertical;
- 140 células: quatorze na vertical e dez na horizontal;
- Em todos os casos, as células possuíam o mesmo tamanho e formato retangular.

Para cada configuração de *Grid*, o mapeamento foi iniciado através da opção *Start Mapping* no *menu MAP*. O *timer* utilizado no mapeamento foi configurado com intervalos diferentes para cada configuração de *Grid*, sendo:

- 1ª configuração (Vinte células): mapeamento com *timer* de duração de um minuto e intervalo de coleta de oito segundos, totalizando sete coletas;
- 2ª configuração (49 células): mapeamento com *timer* de duração de dois minutos e intervalo de coleta de dezesseis segundos, totalizando sete coletas;
- 3ª configuração (140 células): uma primeira sequência de mapeamento com *timer* de duração de vinte segundos e intervalo de quatro segundos, totalizando cinco coletas. Uma segunda sequência de mapeamento com *timer* de duração de 80 segundos e intervalo de onze segundos, totalizando sete coletas;
- Em todos os casos as opções *Timer Save* e *Timer Send* foram marcadas, de modo que todas as imagens e arquivos de texto dos mapas fossem salvos na memória interna do *smartphone*, e todos os arquivos de texto dos mapas fossem enviados para o *notebook* via *Bluetooth*, já que este além de controlar a aplicação, também se conectou à ela para recebimento dos mapas e avaliação desta funcionalidade;
- Durante o mapeamento, o robô *E-Puck* foi deslocado pelo campo manualmente algumas vezes, de modo a se testar a identificação dinâmica do robô.

Os resultados do mapeamento obtidos em cada configuração do *Grid* são apresentados nas Tabelas 6.1 à 6.4. Os resultados são mensurados em número de objetos identificados em cada configuração utilizada. Além disso, cada coleta representa exatamente um mapeamento

por *frame* capturado. A Tabela 6.5 apresenta a média e desvio geral da média de acertos para as três configurações.

Tabela 6.1. Resultado da 1ª configuração: vinte células, sete coletas (O autor)

Coleta	Objetos identificados					
Número	Obstáculos (qtd)	Obstáculos (acerto)	Objetivo (qtd)	Objetivo (acerto)	*E-Puck* (qtd)	*E-Puck* (acerto)
1	8	88,89%	1	100%	1	100%
2	7	77,78%	1	100%	1	100%
3	8	88,89%	1	100%	1	100%
4	7	77,78%	1	100%	1	100%
5	6	66,67%	1	100%	1	100%
6	9	100%	1	100%	1	100%
7	9	100%	1	100%	1	100%
Média de acerto		85,72%		100%		100%
Desvio		11,45%		0%		0%

Tabela 6.2. Resultado da 2ª configuração: 49 células, sete coletas (O autor)

Coleta	Objetos identificados					
Número	Obstáculos (qtd)	Obstáculos (acerto)	Objetivo (qtd)	Objetivo (acerto)	*E-Puck* (qtd)	*E-Puck* (acerto)
1	8	88,89%	1	100%	1	100%
2	9	100%	1	100%	1	100%
3	9	100%	1	100%	1	100%
4	8	88,89%	1	100%	1	100%
5	7	77,78%	1	100%	1	100%
6	6	66,67%	1	100%	1	100%
7	6	66,67%	1	100%	1	100%
Média de acerto		84,13%		100%		100%
Desvio		13,09%		0%		0%

Tabela 6.3. Resultado da 3ª configuração, 1ª sequência de mapeamento: 140 células, quatro das cinco coletas (O autor)

Coleta	Objetos identificados					
Número	Obstáculos (qtd)	Obstáculos (acerto)	Objetivo (qtd)	Objetivo (acerto)	*E-Puck* (qtd)	*E-Puck* (acerto)
1	8	88,89%	1	100%	1	100%
2	9	100%	1	100%	1	100%
3	7	77,78%	1	100%	1	100%
4	8	88,89%	1	100%	1	100%
Média de acerto		88,89%		100%		100%
Desvio		7,86%		0%		0%

Tabela 6.4. Resultado da 3ª configuração, 2ª sequência de mapeamento: 140 células, sete coletas (O autor)

Coleta	Objetos identificados					
Número	Obstáculos (qtd)	Obstáculos (acerto)	Objetivo (qtd)	Objetivo (acerto)	*E-Puck* (qtd)	*E-Puck* (acerto)
1	5	55,56%	1	100%	1	100%
2	8	88,89%	1	100%	1	100%
3	6	66,67%	1	100%	1	100%
4	6	66,67%	1	100%	1	100%
5	8	88,89%	1	100%	1	100%
6	6	66,67%	1	100%	1	100%
7	8	88,89%	1	100%	1	100%
Média de acerto		74,61%		100%		100%
Desvio		12,89%		0%		0%

Tabela 6.5. Média e desvio geral da média de acertos para as três configurações (O autor)

Configuração	Objetos identificados		
	Obstáculos (média acerto)	Objetivo (média acerto)	*E-Puck* (média acerto)
1ª	85,72%	100%	100%
2ª	84,13%	100%	100%
3ª (primeira sequência)	88,89%	100%	100%
3ª (segunda sequência)	74,61%	100%	100%
Média de acerto	83,34%	100%	100%
Desvio	5,32%	0%	0%

Para complemento dos resultados, um exemplo de mapeamento, contendo imagem do mapa e arquivo de texto gerados, é apresentado nas Figuras 6.3 e 6.4, respectivamente. Este exemplo é referente à sexta coleta da 1ª configuração.

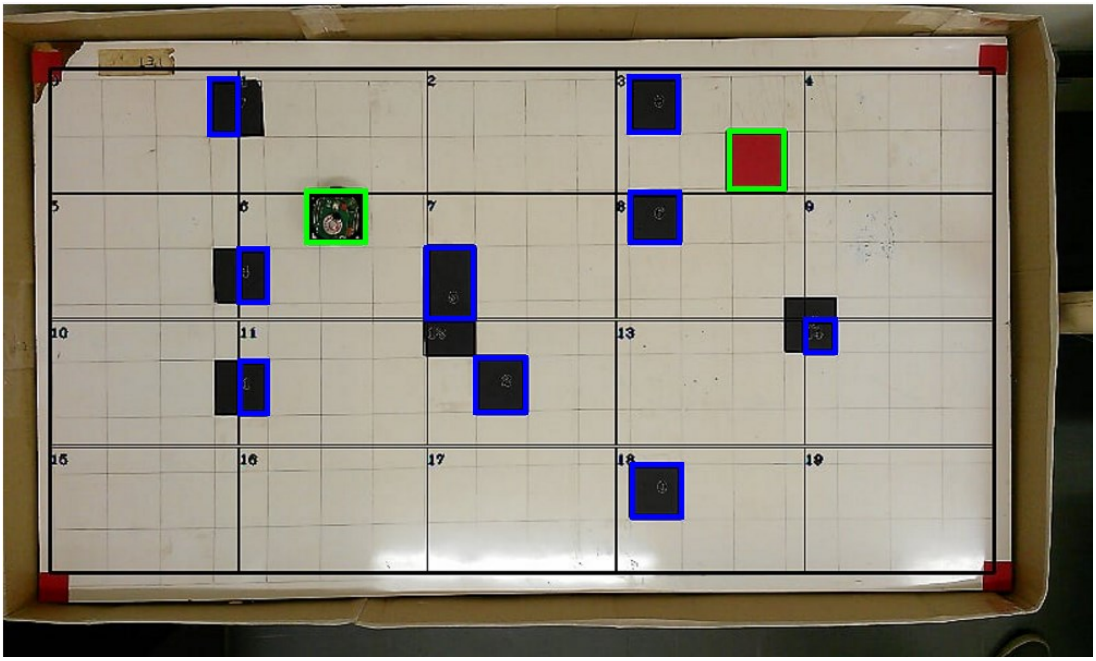


Figura 6.3. Imagem do mapa gerado na sexta coleta da 1ª configuração (O autor)

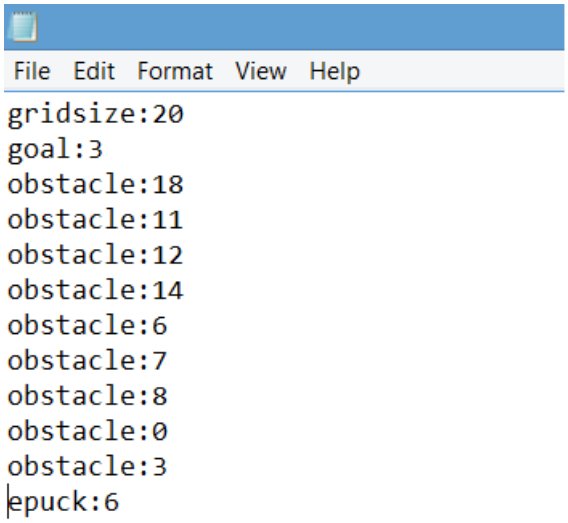


Figura 6.4. Arquivo de texto gerado na sexta coleta da 1ª configuração (O autor)

Observa-se na Figura 6.4, o arquivo de texto do mapeamento na forma, *estado:número_da_célula*, como já explicado anteriormente. Além disso, a primeira linha do arquivo contém o tamanho do *grid*, ou seja, a quantidade de células em que o campo foi dividido, no caso, vinte. Como exemplificação, a célula seis contém o estado *epuck*, confirmado na Fig. 6.3, onde o *E-Puck* se encontra contornado por um retângulo verde.

6.2 Discussão

Antes da discussão dos resultados, após apresentação da aplicação e das suas funcionalidades, cabe uma breve justificativa sobre a utilização do conceito de visão computacional neste trabalho. Apesar de a aplicação desenvolvida limitar-se a busca de informações pré-estabelecidas nas imagens capturadas pela câmera do dispositivo *Android* através de funções de filtragem e segmentação de contorno, caracterizando, a princípio, um PID, como apresentado no capítulo II, os processos aqui descritos serão tratados como pertencentes a área de visão computacional. Isso ocorre porque a aplicação *Android*, com base nas informações extraídas, é capaz de fornecer um significado para a imagem ou imagens capturada(s) através da formulação de um mapa com informações objetivas, no caso, o posicionamento dos obstáculos, objetivo e robôs *E-Puck* presentes na mesa (ou

campo), podendo também transmitir essas informações para os robôs via comunicação *Bluetooth*.

Com relação aos resultados obtidos, nota-se primeiramente através das Figuras 6.2 e 6.3 que a imagem do campo está ligeiramente distorcida. O campo foi construído como um retângulo de dimensões aproximadamente exatas, contudo, este é visto ligeiramente alongado, com lados não exatamente paralelos entre si. Este efeito de distorção é causado pela lente da câmera ao projetar uma cena em três dimensões do ambiente real, em um *sensor* de luz bidimensional (JEPSON, 2011).

Os efeitos de distorção causados pela câmera também podem ser observados na região de interesse do campo virtual formado, contornado de preto na Fig. 6.3, onde este não coincide exatamente com todos os centros dos marcadores em vermelho. Porém, apesar da distorção, a aplicação obteve bons resultados. Nos testes realizados, o campo virtual formado pelos marcadores foi encontrado em 100% das vezes, englobando todos os objetos em seu interior, após os parâmetros no modo de operação *Crop and Goal* terem sido ajustados. Deste modo, optou-se por uma abordagem sem calibração da câmera neste primeiro trabalho, sugerindo o processo de calibração para trabalhos futuros.

Os resultados apresentados nas Tabelas 6.1 à 6.4 destacam o fato de que juntamente com o campo, o objetivo em vermelho também foi encontrado em todas as vezes, produzindo uma média de acerto de 100%. Com relação ao *E-Puck*, os testes mostraram que a taxa de acerto na identificação do *E-Puck* foi de 100% em todos os casos. Contudo, observa-se na Tab. 6.3 que a quinta coleta, na 3ª configuração, 1ª sequência de mapeamento, foi desconsiderada. Na quinta coleta foi observado um *outlier*, ou seja, um valor atípico, na identificação do *E-Puck*, onde este não foi encontrado. Este erro na identificação ocorreu durante o deslocamento do *E-Puck* manualmente, onde o robô foi coberto pela mão do usuário e não identificado, inviabilizando esta coleta.

Por sua vez, a identificação dos obstáculos esteve sujeita à uma significativa variação. Observa-se na Tabela 6.5 que a média de acerto das médias de acerto nas três configurações ficou em torno de 83,34% com um desvio padrão de 5,32%. Essa variação foi causada, particularmente, pelos efeitos da iluminação presente no ambiente, assim como observado no trabalho de (SUMBAL, 2016). Estes efeitos englobam desde a iluminação não uniforme sobre o campo, observada com detalhe nos pontos de luz nas partes inferior e direita da Fig. 6.3, além do efeito de variação na iluminação causado pela frequência de oscilação da luz no ambiente, no caso, da frequência de oscilação da lâmpada do tipo fluorescente presente no laboratório.

Não obstante, observou-se uma leve variação na identificação dos obstáculos conforme o aumento da quantidade de células no campo. Isso provavelmente ocorreu pois, quanto maior o número de células, maior o número de intersecções dos objetos com as mesmas. Como o algoritmo de mapeamento foi projetado para identificar a intersecção com maior área, quando existem várias intersecções de mesma área, e sendo estas as maiores intersecções, pode ocorrer uma variabilidade na definição do estado da célula, onde por ora o objeto pode ser considerado como pertencente à uma célula, se esta intersecção for encontrada primeiro, e em outro momento, pertencente à uma célula ao lado. Essa variabilidade pode afetar significativamente o estado das células na região do objeto, mesmo que este esteja estático. Esta é uma característica do algoritmo de mapeamento empregado, e pode ser reestruturada em trabalhos futuros. Neste sentido, há de se mencionar que a aplicação optou por um mapeamento com identificação direta do estado da célula ao se encontrar qualquer objeto pertencente à esta célula por meio da intersecção da área do objeto com a célula. Esta estruturação de mapeamento gera um inconveniente em algumas situações, como observado no arquivo de texto de mapeamento na Fig. 6.4. Nota-se que a sexta célula possui dois estados, *epuck* e *obstacle*. Cada célula deve possuir apenas um estado, dessa forma, para um trabalho futuro, uma avaliação deve ser realizada de modo que durante o mapeamento, todos os objetos em uma mesma célula sejam avaliados e se escolha o objeto com a maior área dentro da célula, atribuindo este objeto como o estado final da célula.

Apesar da distorção da câmera e da influência da iluminação do ambiente, a aplicação gera mapas com boa confiabilidade, especialmente com relação à identificação do *E-Puck* e do objetivo, além de possuir uma taxa de acerto maior do que 80% na identificação dos obstáculos, como apresentado na Tab. 6.5. Além disso, a aplicação é totalmente modular, permitindo que novas formas de mapeamento sejam implementadas e novos meios de filtragem dos objetos sejam definidos, de modo que os efeitos negativos do ambiente sejam levados em consideração e tratados, sem que as demais funcionalidades da aplicação sejam perturbadas. Por fim, a aplicação realiza o mapeamento com uma boa taxa de confiabilidade e fornece meios flexíveis de configuração em cada etapa do mapeamento.

CAPÍTULO VII

CONCLUSÃO

O mercado da robótica cresce em um ritmo acelerado no mundo todo, com destaque para a robótica móvel. Robôs móveis necessitam, além da percepção do ambiente, de um planejamento de trajetória, de modo que o melhor trajeto seja definido para realização de suas atividades. Melhor trajeto, refere-se à um trajeto onde o robô possa sair de um ponto inicial e atingir um ponto final, seu objetivo, com a menor distância percorrida e livre de obstáculos. Estudos com planejamento de trajetória são dispendiosos em algumas situações onde se deseja realizar os testes dos algoritmos implementados em um ambiente real, seja devido ao alto custo dos equipamentos ou de utilização do ambiente. Geralmente, simulações são realizadas com robôs em ambientes virtuais, fornecendo resultados bastante satisfatórios. Contudo, a realização de simulações e os testes em ambiente real para verificação dos resultados obtidos em simulação, constituem o par ideal em um estudo ou desenvolvimento de aplicações robóticas. Assim, este trabalho propôs o desenvolvimento de uma aplicação *Android* para mapeamento de uma bancada de testes em um ambiente real com obstáculos, objetivo e robôs *E-Puck*, de modo que estudos de planejamento de trajetória possam ser realizados, especialmente baseados em métodos de decomposição celular.

Foi utilizado um aparelho *smartphone* para execução da aplicação *Android*. Este aparelho foi colocado sobre a bancada de testes e foi controlado remotamente por um *notebook* de forma que o campo sobre a bancada contendo os objetos, fosse mapeado. O mapeamento se deu através de técnicas de visão computacional utilizando-se a câmera do dispositivo *Android*, um *smartphone*. Para isso, o campo foi delineado com marcadores em vermelho, bem como todos os outros objetos foram identificados com cores específicas para facilitar a sua identificação. Assim, o algoritmo desenvolvido capturou as imagens da câmera e os processou. O processamento para identificação do campo, do objetivo e dos obstáculos

se deu através de técnicas de filtragem de cor e de identificação de contornos na imagem filtrada. Com o contorno dos objetos identificados, estes foram aproximados por polígonos, mais especificamente, retângulos, que é a forma original destes objetos em campo. No caso do robô *E-Puck* este foi identificado utilizando-se a Transformada circular de *Hough*, e depois envolto em um retângulo para se manter a consistência dos objetos identificados e futura comparação com as células, de formato retangular, em que o campo foi dividido. A divisão do campo em células foi dada pela quantidade de células desejadas horizontalmente e verticalmente, informadas pelo usuário da aplicação. O usuário também foi o responsável pela configuração de todos os parâmetros essenciais para filtragem de cor e identificação do *E-Puck*. Com os objetos identificados e o campo dividido em células, o mapeamento pôde ser realizado através de um *menu* específico da aplicação. Para cada mapeamento, uma imagem foi gerada destacando a posição de cada objeto em sua respectiva célula, sendo este mapeamento visualizado pelo usuário na tela do dispositivo. Além disso, uma imagem do mapeamento e um arquivo de texto com as informações do mapa foram gerados. O arquivo de texto descrevendo um mapeamento padrão é o que de fato traduz a imagem mapeada para uma forma que possa ser utilizada e interpretada por outros dispositivos, como o próprio robô *E-Puck*. O intuito é que uma trajetória a ser percorrida pelo robô possa ser definida a partir das informações contidas neste arquivo de texto. Porém, como observado no capítulo VI, uma avaliação futura deve ser realizada de modo que durante o mapeamento, todos os objetos em uma mesma célula sejam avaliados e se escolha o objeto com a maior área dentro da célula, atribuindo este objeto como o estado final da célula. Deste modo, o mapeamento não atribuirá mais de um estado à mesma célula. Estas novas estruturas de mapeamento podem ser implementadas conforme necessário a um determinado estudo sendo realizado com a aplicação. A implementação é modular, podendo ser realizada diretamente no corpo da função *mapField*, sem afetar o restante da aplicação.

Além da tarefa essencial de mapeamento fornecida pela aplicação, esta permite que os mapas gerados e arquivos de configurações da aplicação possam ser salvos na memória interna do dispositivo *Android*, e carregados a partir dela também. É possível ainda que clientes remotos se conectem à aplicação *Android* via protocolo *Bluetooth*, funcionando esta como um servidor *Bluetooth* para envio dos mapas gerados aos clientes conectados à aplicação. Para coleta de dados e testes, implementou-se ainda uma funcionalidade baseada em *timer* (ou temporizador) que permite que arquivos de mapas sejam salvos e/ou enviados via *Bluetooth* de forma automática a cada intervalo de tempo definido pelo usuário.

Os testes realizados demonstraram que a aplicação é bastante flexível e modular com relação à configuração e implementação das técnicas utilizadas no mapeamento. Testes

realizados apresentaram taxas de acerto superior a 80% na identificação dos obstáculos, e de 100% na delimitação do campo, na identificação do objetivo e do robô *E-Puck*. Contudo, efeitos de variação na iluminação ambiente podem ser tratados pela aplicação, de modo que a identificação dos obstáculos atinja taxas superiores às conseguidas neste trabalho. Além disso, apesar de ter pouco efeito negativo durante o mapeamento, a correção da distorção das imagens produzidas pela câmera pode ser realizada em trabalhos futuros. Felizmente, esta distorção é uma constante da câmera utilizada, e pode ser encontrada e utilizada para calibração e remapeamento dos pixels da imagem. Para isso, uma matriz chamada, matriz de projeção, é utilizada no remapeamento dos pixels. Os detalhes dessa operação de calibração e remapeamento podem ser encontrados na referência da biblioteca *OpenCV*, no tópico *Camera Calibration and 3D Reconstruction* (OPENCV, 2017b).

REFERÊNCIAS BIBLIOGRÁFICAS

ANDROID. Develop Apps. **Android Developers**, 2017. Disponivel em: <https://developer.android.com/develop/index.html>. Acesso em: 05 jul. 2017.

APPLE. Color Management Overview. **Developer - Guides and Sample Code**, 2005. Disponivel em: <https://developer.apple.com/library/content/documentation/GraphicsImaging/Conceptual/csintro/csintro_colorspace/csintro_colorspace.html>. Acesso em: 28 jun. 2017.

BALLARD, D. H. Generalizing the Hough Transform to detect arbitrary shapes. **Pattern Recognition**, v. 13, n. 2, p. 111-122, 1981. Disponivel em: <http://masters.donntu.org/2012/iii/lidke/library/ballard.pdf>.

BARROS, T. O que é smartphone e para que serve? **techtudo**, 2012. Disponivel em: <http://www.techtudo.com.br/artigos/noticia/2011/12/o-que-e-smartphone-e-para-que-serve.html>. Acesso em: 08 jul. 2017.

BEHRING, C. et al. **An Algorithm for Robot Path Planning with Cellular Automata**. Proceedings of the Fourth International Conference on Cellular Automata for Research and Industry: Theoretical and Practical Issues on Cellular Automata. Karlsruhe: Springer-Verlag London. 2000. p. 11-19.

BERGER, N. Amazon rolls out Kiva robots for holiday season onslaught. **REUTERS**, 2014. Disponivel em: <http://uk.reuters.com/article/us-amazon-com-kiva-idUSKCN0JF1NF20141201>. Acesso em: 07 jul. 2017.

CARDANI, D. Adventures in HSV Space. **RobotLab**, 2007. Disponivel em: <http://robotlab.itk.ppke.hu/~rakadam/hsvspace.pdf>. Acesso em: 28 jun. 2017.

COPPELIA ROBOTICS. V-rep, virtual robot experimentation platform. **Coppelia Robotics**, 2017. Disponivel em: <http://www.coppeliarobotics.com/>. Acesso em: 08 jul. 2017.

CORKE, P. **Robotics, Vision and Control:** Fundamental Algorithms in MATLAB. 1. ed. Berlin-Heidelberg: Springer-Verlag, 2011. 570 p.

CYBERBOTICS LTD. Webots: robot simulator. **Cyberbotics**, 2017. Disponivel em: <https://www.cyberbotics.com/overview>. Acesso em: 08 jul. 2017.

EPFL. E-puck education robot. **e-puck**, 2017. Disponivel em: <http://www.e-puck.org/>. Acesso em: 08 jul. 2017.

FERGUS, R. Image Filtering. **Computer Vision Lecture - New York University**, 2016. Disponivel em: <http://cs.nyu.edu/~fergus/teaching/vision/3_filtering.pdf>. Acesso em: 29 jun. 2017.

FERREIRA, G. B. S.; VARGAS, P. A.; OLIVEIRA, G. M. B. **An Improved Cellular Automata-Based Model for Robot Path-Planning**. Advances in Autonomous Robotics Systems. Birmingham: Springer. 2014. p. 25-36.

FLUMINENSE, U. F. Matrices and Binary Images. **Mathematics:** numbers and operations, 2017. Disponivel em: <http://www.uff.br/cdme/matrix/matrix-html/matrix_boolean/matrix_boolean_en.html>. Acesso em: 29 jun. 2017.

GANGULY, N. et al. BISON Project - A Survey on Cellular Automata. **Department of Computer Science and Engineering - Unibo**, 2003. Disponivel em: <http://www.cs.unibo.it/bison/publications/CAsurvey.pdf>. Acesso em: 07 jul. 2017.

GLENIOSP. OCVCmakeTCCFinal. **GitHub**, 2017. Disponivel em: <https://github.com/GlenioSP/OCVCmakeTCCFinal>. Acesso em: 09 jul. 2017.

GONZAGA, A. Aula 7 - Representação e Descrição de Estruturas Bi-dimensionais. **SEL 5886-Disciplina de Visão Computacional - DEE USP São Carlos**, 2017. Disponivel em: <http://iris.sel.eesc.usp.br/sel886/Aula_7.pdf>. Acesso em: 18 jul. 2017.

GRADLE INC. Gradle Build Tool. **Gradle**, 2017. Disponivel em: <https://gradle.org/>. Acesso em: 05 jul. 2017.

GSMARENA. Samsung I9190 Galaxy S4 mini. **GSMARENA**, 2013. Disponivel em: <http://www.gsmarena.com/samsung_i9190_galaxy_s4_mini-5375.php>. Acesso em: 11 jul. 2017.

HIGA, P. 95,5% dos smartphones vendidos no Brasil são Androids. **tecnoblog**, 2016. Disponivel em: <https://tecnoblog.net/203749/android-ios-market-share-brasil-3t-2016/>. Acesso em: 05 jul. 2017.

INSTRUMENTS, NATIONAL. Spatial Filtering. **NI Vision 2011 Concepts Help**, 2011. Disponivel em: <http://zone.ni.com/reference/en-XX/help/372916L-01/nivisionconcepts/spatial_filtering/>. Acesso em: 29 jun. 2017.

JEPSON, A. Image Projection. **Introduction to Image Understanding - Department of Computer Science, University of Toronto**, 2011. Disponivel em: <http://www.cs.toronto.edu/~jepson/csc420/notes/imageProjection.pdf>. Acesso em: 20 jun. 2017.

JETBRAINS. Code Iris. **JetBrains PLUGINS**, 2017. Disponivel em: <https://plugins.jetbrains.com/plugin/7324-code-iris>. Acesso em: 18 jul. 2017.

KAEHLER, A.; BRADSKI, G. **Learning OpenCV 3:** COMPUTER VISION IN C++ WITH THE OPENCV LIBRARY (e-Book version). 1. ed. Sebastopol, CA: O'Reilly Media, 2017. 1255 p.

KHOSO, M.; KHAN, K. Smartphones: A Supercomputer in Your Pocket. **Northeastern University**, 2016. Disponivel em: <http://www.northeastern.edu/levelblog/2016/04/21/smartphones-supercomputer-in-your-pocket/>. Acesso em: 08 jul. 2017.

KITWARE. CMake. **CMake**, 2017. Disponivel em: <https://cmake.org/>. Acesso em: 05 jul. 2017.

KUKA. Automation in the automotive industry. **KUKA**, 2017. Disponivel em: <https://www.kuka.com/en-cn/industries/automotive>. Acesso em: 07 jul. 2017.

LEADRIEN. opencv_native_androidstudio. **GitHub**, 2017. Disponivel em: <https://github.com/leadrien/opencv_native_androidstudio>. Acesso em: 05 jun. 2017.

MATHWORKS. imgaussfilt (function) - Smooth image with Gaussian filter. **MathWorks Documentation**, 2017. Disponivel em: <https://www.mathworks.com/help/images/ref/imgaussfilt.html>. Acesso em: 29 jun. 2017.

OLIVEIRA, G. M. B.; VARGAS, P. A.; FERREIRA, G. B. S. **Investigating a cellular automata model that performs three distance diffusion on a robot path planning**. Proceedings of the European Conference on Artificial Life 2015. York: THE MIT PRESS. 2015. p. 271–278.

OPENCV. OpenCV Library. **OpenCV**, 2017a. Disponivel em: <http://opencv.org/>. Acesso em: 04 jul. 2017.

OPENCV. OpenCV API Reference. **OpenCV**, 2017b. Disponivel em: <http://docs.opencv.org/3.0-beta/modules/refman.html>. Acesso em: 02 jul. 2017.

RHODY, H. Lecture 10: Hough Circle Transform. **Introduction to Digital Image Processing - RIT Center for Imaging Science**, 11 out. 2005a. Disponivel em:

<https://www.cis.rit.edu/class/simg782/lectures/lecture_10/lec782_05_10.pdf>. Acesso em: 02 jul. 2017.

RHODY, H. Lecture 9: Object Description and Location - Hough Transform. **Introduction to Digital Image Processing - RIT Center for Imaging Science**, 2005b. Disponivel em: <https://www.cis.rit.edu/class/simg782/lectures/lecture_09/lec782_05_09.pdf>. Acesso em: 03 jul. 2017.

SHAPIRO, L. G.; HARALICK, R. M. Image Segmentation Techniques. **COMPUTER VISION, GRAPHICS, AND IMAGE PROCESSING**, Ann Arbor, v. 29, p. 100-132, 1985.

SHEAD, S. Amazon now has 45,000 robots in its warehouses. **Business Insider**, 2017. Disponivel em: <http://uk.businessinsider.com/amazons-robot-army-has-grown-by-50-2017-1>. Acesso em: 07 jul. 2017.

SIANO, G.; GLIELMO, L. **Low-Cost Mobile Robotics Platform for Active**. 20th IMEKO TC4 International Symposium and 18th International Workshop on ADC Modelling and Testing. Benevento: [s.n.]. 2014.

SIEGWART, R.; NOURBAKHSH, I. R. **Introduction to Autonomous Mobile Robots**. Cambridge, MA: The MIT Press, 2004. 335 p.

SUMBAL, M. S. Environment Detection and Path Planning Using the E-puck Robot. **International Research Journal of Engineering and Technology (IRJET)**, v. 3, n. 1, p. 151-160, jan 2016.

SYED, U. A.; KUNWAR, F. Cellular Automata Based Real-time Path-planning for Mobile Robots. **International Journal of Advanced Robotic Systems**, Thousand Oaks, CA, v. 11, n. 7, 2014.

SZELISKI, R. **Computer Vision:** Algorithms and Applications (Draft version). 1. ed. [S.l.]: Springer, 2010. 979 p.

UNIVERSE, N. I. T. The Electromagnetic Spectrum. **IMAGINE THE UNIVERSE!**, 2013. Disponivel em: <https://imagine.gsfc.nasa.gov/science/toolbox/emspectrum1.html>. Acesso em: 28 jun. 2017.

VANIAN, J. The Multi-Billion Dollar Robotics Market Is About to Boom. **FORTUNE**, 2016. Disponivel em: <http://fortune.com/2016/02/24/robotics-market-multi-billion-boom/>. Acesso em: 07 jul. 2017.

VISWANATHAN, R.; KRISHNAN, M. H. A New Concept of Reduction of Gaussian Noise in Images Based on Fuzzy Logic. **Applied Mathematical Sciences**, Ruse, v. 7, n. 12, p. 595 - 602, out. 2013. Disponivel em: <http://www.m-hikari.com/ams/ams-2013/ams-9-12-2013/harikrishnanAMS9-12-2013.pdf>. Acesso em: 29 jun. 2017.

WOODS, R. E.; GONZALEZ, R. C. **Digital Image Processing**. 3. ed. Upper Saddle River, New Jersey: Pearson Prentice Hall, 2008. 954 p.

ZURIARRAIN, J. M. Android já é o sistema operacional mais usado do mundo. **Edição Brasil do EL PAÍS**, 2017. Disponivel em: <https://brasil.elpais.com/brasil/2017/04/04/tecnologia/1491296467_396232.html>. Acesso em: 05 jul. 2017.