UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Ricardo Camilo Machado

SISTEMA DE VISÃO COMPUTACIONAL PARA MAPEAMENTO DE OBSTÁCULOS EM UMA MESA DE PROVAS DE ROBÔS

Uberlândia, Brasil

2017

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Ricardo Camilo Machado

SISTEMA DE VISÃO COMPUTACIONAL PARA MAPEAMENTO DE OBSTÁCULOS EM UMA MESA DE PROVAS DE ROBÔS

Trabalho de conclusão de curso apresentado à Faculdade de Engenharia Mecânica da Universidade Federal de Uberlândia, Minas Gerais, como requisito exigido parcial à obtenção do grau de Bacharel em Engenharia Mecatrônica.

Orientador: Prof. Dr. Mauricio Cunha Escarpinati

Universidade Federal de Uberlândia – UFU

Faculdade de Engenharia Mecânica

Bacharelado em Engenharia Mecatrônica

Uberlândia, Brasil

2017

Agradecimentos

Começo essa dedicatória com a seguinte pergunta, porque é tão difícil agradecer? Não falo do agradecimento em si pois o sentimento de reconhecimento já existe, mas sim da dificuldade de expressar essa gratidão sem recorrer aos famosos clichês. Penso que é um sentimento muito complexo de se traduzir, um sentimento que se recusa a se fazer palavra, mas tenho certeza que há um bocado de pessoas que realmente fazem valer a pena essa tentativa. Assim agradeço:

A Deus por ter colocado pessoas essenciais na minha vida, me dado saúde e por ter "mexido alguns pauzinhos"para que a realização deste trabalho fosse possível.

Aos meus pais, Soni Camilo Gomes Filho e Lusmar Machado Borges, que me forneceram toda a base necessária, todo o amor e que acreditaram em mim quando ninguém mais acreditava.

Aos meus irmãos Caroline Camilo Machado e Alexsander Camilo Machado que foram meus companheiros de jornada que me dão tantas alegrias e seguiram comigo nessa estrada me dando força e carinho.

A minha namorada Letícia Menezes Gonçalves, que tem sido o meu porto seguro, sempre ao meu lado me incentivando, me dando força e carinho.

Aos meus amigos que forneceram bons conselhos e materiais que foram essenciais para o desenvolvimento deste projeto. Em particular ao Bruno Kamada que me emprestou seu notebook para o desenvolvimento dos algoritmos.

Ao meu orientador Prof. Dr. Maurício Cunha Escarpinati que com o seu conhecimento forneceu conselhos valiosos e que com a sua paciência me deu todo o apoio necessário.

Aos coordenadores do curso de engenharia mecatrônica, em espcial a Profa. Dra. Vera Lúcia D. S. Franco, sem o apoio deles esse trabalho não teria sido realizado.

'Nós devemos saber, Nós saberemos.'

David Hilbert

Resumo

O processo de extração de informações do meio que permitem identifica-lo e analisa-lo via algoritmos de visão computacional é parte essencial do desenvolvimento de qualquer projeto que visa o treinamento de robôs. Nesta etapa são geradas informações a respeito dos obstáculos que os robôs possam vir a se deparar, sobre os limites do seu espaço de trabalho, informações a respeito sobre os próprios robôs, que podem desenvolver relações de trabalho específicas dado que cada um tem sua função no meio em questão e por fim informações sobre o objetivo, que de certa forma da sentido a existência do robô, lhe fornece um propósito específico dentro do ambiente. Sendo assim este trabalho se propõe a implementar novas funções a uma aplicação já existente *OCVCmakeTCCFinal* de (PIMENTEL et al., 2017), ampliando a sua capacidade de reconhecer não só quantitativamente como qualitativamente os robôs *E-Puck* identificando suas cores bem como a direção de seus deslocamentos, para isso foram desenvolvidos uma série de artifícios que permitissem aos algoritmos de segmentação reconhecer tais atributos. Todos os algoritmos foram desenvolvidos fazendo o uso da biblioteca de visão computacional *OpenCV* bem como do framework *JNI* que permiti a integração de código C++ em aplicações *Android*. Foi desenvolvida uma mesa de teste de forma semelhante aquela utilizada por (PIMENTEL et al., 2017) com base nos mesmos elementos utilizados por ele, com exceção dos *E-Pucks* que foram modelados com base em revestimentos cilíndricos coloridos. Foram executadas duas baterias de testes onde se variou alguns parâmetros tais como a tonalidade das cores utilizadas para os revestimentos entre outros atributos afim de melhorar a resposta da aplicação aos elementos do meio.

Palavras-chave: *Visão Computacional. E-Puck Android. OpenCV. JNI.*

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

θ – ângulo das equações paramétricas da Transformada Circular de Hough

ARM – Advanced RISC Machine

FACOM – Faculdade de Computação

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

IBM – International Business Machines

IDE – Integrated Development Environment

NDK – Native Development Kit

OpenCV – Open Source Computer Vision Library

pixel – picture element

PDI – Processamento Digital de Imagens

PVC – Policloreto de vinila

SDK – Standard Development Kit

SO – Sistema Operacional

UFU – Universidade Federal de Uberlândia

Sumário

1 Introdução

A robótica móvel é uma disciplina que se preocupa em projetar o hardware e o software de tal forma que os robôs possam realizar sua tarefa na presença de ruído, informações de sensores contraditórias e inconsistentes e possivelmente em ambientes dinâmicos (ambientes que alteram a sua estrutura com o passar do tempo). Os robôs móveis podem ser controlados remotamente, guiados por ambientes especialmente projetados ou totalmente autônomos, ou seja, independentes de quaisquer conexões com o mundo exterior tomando as decisões por si próprios. Os robôs móveis são amplamente utilizados em aplicações industriais, incluindo transporte, inspeção, exploração, vigilância, cuidados de saúde, robôs de entretenimento ou até guias turísticos de museu. O que os torna interessantes para aplicações científicas é o fato de que eles fecham o laço entre percepção e ação e, portanto, podem ser usados como ferramentas para realizar uma tarefa com comportamento inteligente. O comportamento de um robô móvel não é o resultado da programação do robô sozinho, mas resulta da interação de três componentes fundamentais:

- O programa executado no robô (a "tarefa").
- O hardware físico do robô tais como sensores , os motores, carga da bateria e etc.
- O ambiente com os objetos e obstáculos e objetivos a serem alcançados.

Identificar e analisar o ambiente no qual se pretende que um robô navegue é um dos passos mais importantes quando se inicia um projeto de robótica. A razão é que a capacidade de um robô de realizar seu trabalho com eficiência e segurança é extremamente dependente da compreensão do seu entorno. A navegação dos robôs é uma tarefa complexa que não se baseia apenas em ensinar o que o robô deve saber ou fazer, é um procedimento que requer vigilância constante do ambiente no qual ele está inserido, dado que há um dinamismo inerente ao meio.

Em geral, existem dois tipos de ambientes: estruturados e não estruturados. Entender e identificar em qual tipo de ambiente o robô irá operar é de suma importância. Um ambiente estruturado é essencialmente um espaço definido de forma precisa. Esse tipo de ambiente tem o mínimo possível de variáveis e em teoria é um sistema rígido - o que significa que um robô sabe o que esperar ao navegar por ele o tempo todo. Um ambiente estruturado é previsível. Neste ambiente pode se encontrar algumas variáveis tais como: luminosidade, temperatura, umidade, rugosidade da superfície entre outras, todas difíceis de controlar. O ambiente não estruturado é o contrário, o espaço não é definido precisamente e não é possível prever quando e nem onde um objeto pode estar.

Ao se analisar um ambiente para robôs se faz necessário o emprego de técnicas de processamento digital de imagens, que possam fornecer informações pertinentes a respeito de todos os elementos que modelam o ambiente, tais como: obstáculos, robôs, demarcadores de campo (sinalizadores a partir dos quais se estabelecem os limites do ambiente de teste) e estruturas que podem configurar um objetivo a ser alcançado pelos robôs. Algoritmos de segmentação extraem essas informações do entorno, que serão utilizadas posteriormente para diversos fins, tais como: treinamento de robôs, mapeamento e o cálculo de trajetórias. Basicamente a análise do ambiente tem por meta treinar o robô para que este possa interagir com o meio.

Como o reconhecimento dos elementos que compõem o ambiente é parte essencial de vários processos que ocorrem em diversos segmentos dentro da robótica, este trabalho se propõe a estender as funcionalidades já existentes da aplicação *OCVCmakeTCCFinal* desenvolvida por (PIMENTEL et al., 2017) que foi projetada para mapear um ambiente simulado para robôs *E-Pucks*, destinada a sistemas *Android*, a partir da integração deste com a biblioteca de visão computacional *OpenCV*. Foram Implementadas diversas funções de mapeamento visando identificar os elementos especificamente modelados para o ambiente simulado, esses elementos são: um robô *E-Puck* que tem forma circular, portanto foram utilizados algoritmos baseados na transformada de Hough para reconhecer tal forma, os obstáculos com forma retangular e cor preta, os delimitadores e objetivo também com forma retangular porém de cor vermelha. Assim foram implementados filtros que fazem o uso de algoritmos de segmentação para extrair informações referentes a estes objetos. A aplicação ainda conta com uma função para dividir o campo em diversas células enumeradas, que permitem localizar cada elemento dentro do ambiente.

Fazendo o uso de algoritmos inteligentes e estendo a aplicação desenvolvida por (PIMENTEL et al., 2017) este trabalho se propõe a identificar: um número maior de robôs *E-Pucks*, a direção e sentido do movimento desses robôs e também a cor, que os definem de forma única no entorno. Para isso foram utilizados robôs *E-Puck* bem como revestimentos coloridos afim de conferir-lhes uma identidade dentro do ambiente de testes. Estes revestimentos foram confeccionados de forma a permitir que os algoritmos pudessem reconhecer o sentido do movimento. Foi estudado também a relação entre alguns parâmetros que caracterizam o ambiente de teste e a resposta da aplicação a eles.

1.1 Objetivos

1.1.1 Objetivo Geral

O objetivo do presente projeto é estudar e desenvolver uma nova versão para a aplicação *OCVCmakeTCCFinal* Android desenvolvida por (PIMENTEL et al., 2017) que é uma aplicação de visão computacional que faz o uso da biblioteca OpenCV para o

mapeamento de um ambiente simulado mais precisamente de uma mesa de robôs contendo marcadores, objetivo, obstáculos e robôs *E-Pucks*. Esta nova versão tem por objetivo mapear novas informações a respeito destes robôs, informações essas que são a direção e a cor do revestimento em papel cartolina que foi confeccionado para eles.

1.1.2 Objetivos específicos

Para alcançar o objetivo geral, faz-se necessário estabelecer alguns objetivos específicos, tais como:

- Estudar conceitos de visão computacional, mais especificamente de Processamento Digital de Imagens;
- Estudar técnicas de desenvolvimento para dispositivo móvel que faça uso da plataforma *Android*;
- Implementar técnicas de segmentação e classificação de imagens afim de mapear uma região de testes de robôs inteligentes;
- Estudo sobre técnicas de desenvolvimento de software para plataformas *Android* via *Android Studio* , *SDK tools*, bibliotecas *OpenCV* e o framework *JNI*
- Testes com o sistema projetado.
- Análise e discussão dos resultados obtidos.

1.2 Justificativa

Dado que a análise do ambiente é parte essencial no processo de treinamento dos robôs e que a expansão do mercado concomitantemente com o avanço da tecnologia tornam os robôs produtos mais acessíveis, surge assim a possibilidade de integração desses dispositivos com tecnologias já existentes tais como os smartphones que, configuram uma solução viável e de baixo custo para o mapeamento e análise de ambientes de simulação nos quais, grande parte dos estudos em robótica hoje são realizados. Esses mapeamentos são importantes para o cálculo e planejamento de trajetórias para a orientação desses robôs nos campos de testes. Baseado nisso um projeto que propõe o aperfeiçoamento e extensão de uma aplicação já existente utilizada para o mapeamento justifica o desenvolvimento do mesmo.

1.3 Organização do Trabalho

O trabalho será disposto da seguinte forma:

- **Capítulo 2:** traz uma revisão bibliográfica sobre todos os temas correlatos as necessidades do presente trabalho tais como: visão computacional, representação da imagem digital, espaço de cores, biblioteca *OpenCV*, Sistema Operacional *Android* entre outros.

- **Capítulo 3:** Faz uma explicação detalhada a respeito do funcionamento da aplicação *OCVCmakeTCCFinal* desenvolvida por (PIMENTEL et al., 2017) que é a base para este projeto assim como apresenta parte dos resultados obtidos por ele.

- **Capítulo 4:** Especifica todas condições nas quais foram realizados os teste tais como materiais utilizados, modos de operação para a coleta das informações ponderantes.

- **Capítulo 5:** Discute os resultados encontrados pela realização dos testes.

2 Revisão Bibliográfica

2.1 Introdução

Para o desenvolvimento do trabalho conforme proposto, diversas técnicas e ferramentas são necessárias. Assim sendo, 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.

2.2 Visão Computacional

Visão computacional é o processo de se extrair informações a partir de operações ou transformações feitas em dados tais como imagens digitais, vídeos ou qualquer estrutura de dados multi-dimensional do mundo real que(GARCÍA et al., 2015), pode levar ou a tomada de decisão ou à uma nova representação destes, de modo a descrever o mundo que vemos a nossa volta e reconstruindo suas propriedades, tais como, forma, iluminação e distribuição de cores(SZELISKI, 2010). Analisando do ponto de vista da engenharia é o processo de automatizar as tarefas que o sistema visual humano desempenha, assim como um médico pode identificar um tumor em uma tomografia ou uma pessoa poderia reconhecer o próprio rosto em uma fotografia tais sistemas poderiam fazer o mesmo(SZELISKI, 2010). Dentre algumas aplicações, se pode destacar:

- Reconhecimento ótico de caracteres: leitura de códigos postais escritos a mão, tarefa essa que pode ser executada com o uso de uma rede neural perceptron multicamadas e reconhecimento automático de placas veiculares através do monitoramento do tráfico em tempo real
- Inspeção rápida de peças para a garantia de qualidade das mesmas utilizando uma técnica conhecida como visão estéreo que, pode também ser usada com uma iluminação especializada para medir tolerâncias nas asas da aeronave ou procurando defeitos em peças de aço usando visão de raios-X.
- Modelagem 3D utilizando fotogrametria que é uma técnica utilizada para extrair medidas rigorosas para a construção de modelos em três dimensões a partir de fotografias aéreas.

- Reconhecimento de movimento e gestos humanos aplicação utilizada pela tecnologia Microsoft Kinects para o sistema XBOX.

- Processamento médico de imagens. Tal área é caracterizada pela extração de informação de imagens para realizar diagnósticos sobre os pacientes. Fontes de imagens incluem imagens de microscopia, de radiografia, de angioplastia, de ultrassonografia, de tomografia e de Ressonância magnética(**??**).

- Veículos autônomos, cujo nível de autonomia varia entre total ou parcial, este último usado para somente auxiliar a tarefa de dirigir em situações diversas. A autonomia total usa a visão computacional para a navegação, isto é, para obter a localização, para produzir mapas do ambiente e para detectar obstáculos. Várias montadoras já demonstraram veículos totalmente autônomos, mas tal tecnologia ainda não atingiu maturidade suficiente para estar no mercado. A exploração espacial também está usando veículos autônomos usando a visão computacional, como por exemplo a Mars Exploration Rover da NASA(WIKIPéDIA, 2017).

2.3 A luz

A luz é a região visível do espectro eletromagnético e é a mais familiar porque, como espécie adaptamos receptores (olhos) que são sensíveis à radiação eletromagnética mais intensa emitida pelo Sol, a fonte extraterrestre mais próxima. Os limites do comprimento de onda da região visível vão desde aproximadamente 400 nm (Violeta) até aproximadamente 700 nm (vermelho)(RESNICK; HALLIDAY; WALKER, 1987). Dentro dessa faixa, o olho humano interpreta comprimentos de onda distintos como cores diferentes sendo que a sensibilidade a diferentes comprimentos de onda não é de forma alguma constante, sendo que a maior sensibilidade ocorre aproximadamente aos 555 nm . Podemos definir como monocromáticas as fontes de comprimento único (o que na prática é impossível) e a cor definida por esse comprimento de onda denomina se cor espectral pura, a Figura 1 Apresenta o espectro com destaque para faixa de luz visível.

2.4 Imagem Digital

Uma imagem monocromática é uma função da intensidade bidimensional, denotada por f(x, y), na qual x e y são coordenadas espaciais e aos valores associados ao pares ordenados (x, y) denomina se como intensidade luminosa (brilho) no ponto considerado. Como os computadores não são capazes de processar imagens contínuas, apenas imagens discretizadas, é necessário representar imagens como arranjos bidimensionais de pontos(QUEIROZ; GOMES, 2006).

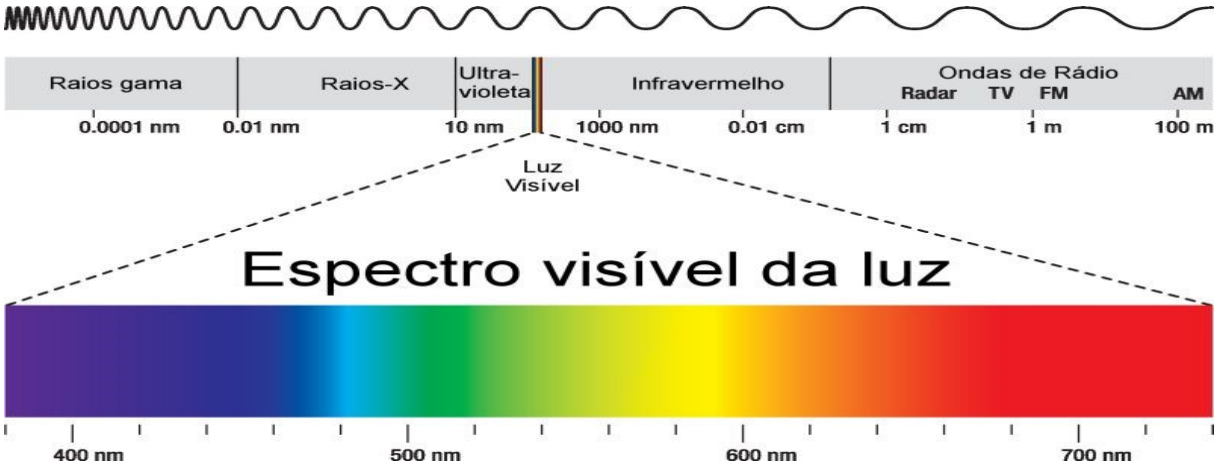

Figura 1 – Espectro da luz

As imagens que as pessoas percebem em atividades visuais corriqueiras consistem de luz refletida dos objetos. A natureza básica de f(x, y) pode ser caracterizada por dois componentes, a saber, a quantidade de incidindo na cena observada e a quantidade de luz refletida pelos objetos da cena. Apropriadamente esses componentes são chamados de iluminação e reflectância, respectivamente, e são representadas por i(x, y) e r(x, y). O produto destas duas funções resulta na imagem representada por f(x, y)(GONZALEZ; WOODS, 2000).

Cada ponto na grade bidimensional que representa a imagem digital é denominado elemento de imagem ou pixel. Na Fig. 3, apresenta-se a notação matricial usual para a localização de um pixel sem arranjo de pixels de uma imagem bidimensional. O primeiro índice denota a posição da linha, m, na qual o pixel se encontra, enquanto o segundo, n,denota a posição da coluna. Se a imagem digital contiver M linhas e N colunas, o índice m variará de 0 a M-1, enquanto n variará de 0 a N-1. Observe-se o sentido de leitura (varredura) e a convenção usualmente adotada na representação espacial de uma imagem digital(GONZALEZ; WOODS, 2000).

2.5 Limiarização

A Limiarização é uma técnica utilizada dentro do contexto da segmentação que se fundamenta na análise da similaridade de níveis de cinza. Dada uma imagem onde se tem um plano de fundo escuro e um conjunto de objetos iluminados sobre este fundo, pode-se a partir da análise de um histograma verificar que os pixels dos objetos e do plano de fundo se agrupam em diferentes intervalos de níveis de cinza estabelecendo assim, entre eles, um limiar ou uma fronteira de decisão(GONZALEZ; WOODS, 2000). Portanto uma maneira de se extrair estes objetos seria a definição de um limiar T. Seja f(x, y) uma imagem contendo apenas um objeto e suponhamos que se f(x, y) > T então este pixel seria

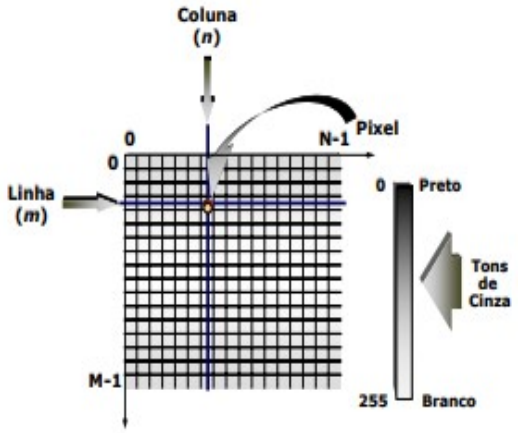

Figura 2 – Representação esquemática de uma imagem digital

pertencente ao conjunto de pixels que definem o objeto e caso contrário seria um ponto do plano de fundo. Assim o processo de segmentação varre toda a imagem comparando cada pixel com este valor T e o classificando como 1 para o objeto e 0 para o plano de fundo, gerando assim uma imagem binária onde fica clara a separação da região que define este objeto. Além da abordagem de limiar único tem-se a de limiar multiníveis onde figura-se mais de um objeto na imagem, definindo assim mais de um limiar como fronteira de separação destes objetos a Figura 3 ilustra os dois tipos de limiares. Geralmente esse tipo de limiarização leva a resultados piores do que a de limiar único(GONZALEZ; WOODS, 2000). A biblioteca OpenCV utilizada neste trabalho e especificada mais detalhadamnete adiante, conta com vários tipos de operações de thresholding gerando diferentes imagens g(x, y) com base no limiar T, dentre elas podemos destacar:

A biblioteca OpenCV utilizada neste trabalho e especificada mais detalhadamnete adiante, conta com vários tipos de operações de thresholding gerando diferentes imagens g(x, y) com base no limiar T, dentre elas podemos destacar:

- Limiarização Binária.
- Limiarização Binária Invertida.
- Limiarização Truncada.
- Limiarização para o zero.
- Limiarização para o zero invertida.

2.6 Espaço RGB

Objetos que emitem luz visível são percebidos em função da soma das cores espectrais emitidas. Tal processo de formação é denominado aditivo. O processo aditivo pode

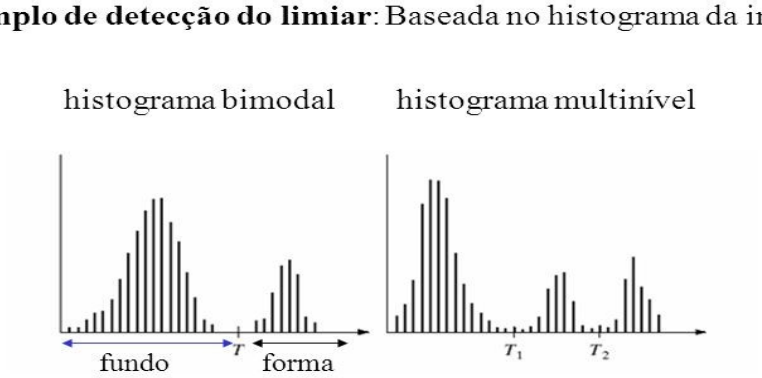

Figura 3 – Exemplos de limiares bimodais e multiníveis

ser interpretado como uma combinação variável em proporção de componentes monocromáticas nas faixas espectrais associadas às sensações de cor verde, vermelho e azul, as quais são responsáveis pela formação de todas as demais sensações de cores registradas pelo olho humano. Assim, as cores verde, vermelho e azul são ditas cores primárias. Este processo de geração suscitou a concepção de um modelo cromático denominado RGB (Red, Green, e Blue) (QUEIROZ; GOMES, 2006), para o qual a Comissão Internacional de Iluminação (CIE) que é uma entidade internacional sem fins lucrativos, estabeleceu os intervalos de comprimento de onda das cores primárias, tal modelo pode ser arranjado em uma geometria cúbica A combinação dessas cores, duas a duas e em igual intensidade, produz as cores secundárias, Ciano, Magenta e Amarelo

2.7 Espaço HSV

O espaço HSV que também é conhecido como HSB (Hue, Saturation e Brightness) é formado por três componentes Hue, Saturation e Value que traduzindo para o português significam Matiz, Saturação e Valor respectivamente. A matiz é a cor propriamente dita, saturação é uma medida de quão pura essa cor é, quanto menor a saturação mais clara ou mais branca essa cor será. A componente valor ou brilho está relacionada a quantidade de brilho ou luminância. Esse sistema foi desenvolvido em meados de 1970 por pioneiros da computação gráfica que trabalhavam na PARC e NYIT e foi formalmente descrito por

Alvy Ray Smith.

Uma das grandes vantagens do sistema HSV em relação aos outros sistemas de cores tais como RGB e CYMK é que do ponto de vista do ser humano ele é mais intuitivo. Sistemas tais como RGB são confusos, por exemplo, imagine um terminal RGB para o qual a cor é controlada por três reguladores de intensidade para cada uma das três cores primárias vermelho, azul e verde e nele se apresenta a cor laranja pura que no sRGB contêm os valores R = 217, G = 118, B = 33 para alterar a saturação desta pela metade devemos diminuir R em 31 unidades, aumentar G em 24 unidades e aumentar B em 59 unidades tal como demonstra a Figura 4 (CONTRIBUTORS, 2017).

Figura 4 – Demonstração das variações nos valores necessárias em cada canal de uma imagem no espaco RGB para manter a mesma cor porem com menos brilho

O espaço de cor HSV pode ser concebido da seguinte forma, temos uma coordenada angular que representa a matiz ou as cores propriamente ditas, temos uma coordenada radial que define a saturação e também um eixo vertical associado a quantidade valor. A partir desta descrição chega se uma geometria cônica que representa o espaço HSV como mostra a Figura 5.

2.8 Biblioteca OPenCV

Desenvolvida inicialmente pela Intel, OpenCV (Open Source Computer Vision) é uma biblioteca multi-plataforma com um forte foco para o processamento digital de imagens em tempo real, escrita em C/C++ que hoje contém interfaces para diversas linguagens tais como Python e Java . Sua primeira versão foi lançada em 2000 e desde então as suas funcionalidades vem sendo desenvolvidas pela comunidade cientifica, ela conta com várias centenas de algoritmos de visão computacional o que torna esta biblioteca uma ferramenta poderosa no campo da visão computacional(BAGGIO, 2015).

Para realização do presente trabalho foi utilizada a interface C++ do OPenCV na qual podemos destacar algumas características que, por estarem ausentes na interface Java, podem conferir algumas vantagens, dentre as quais destaca se:

- Gerenciamento de memória manual: Considerando que a interface Java do OpenCV faz o gerenciamento automático da memória através do coletor de lixo (do inglês

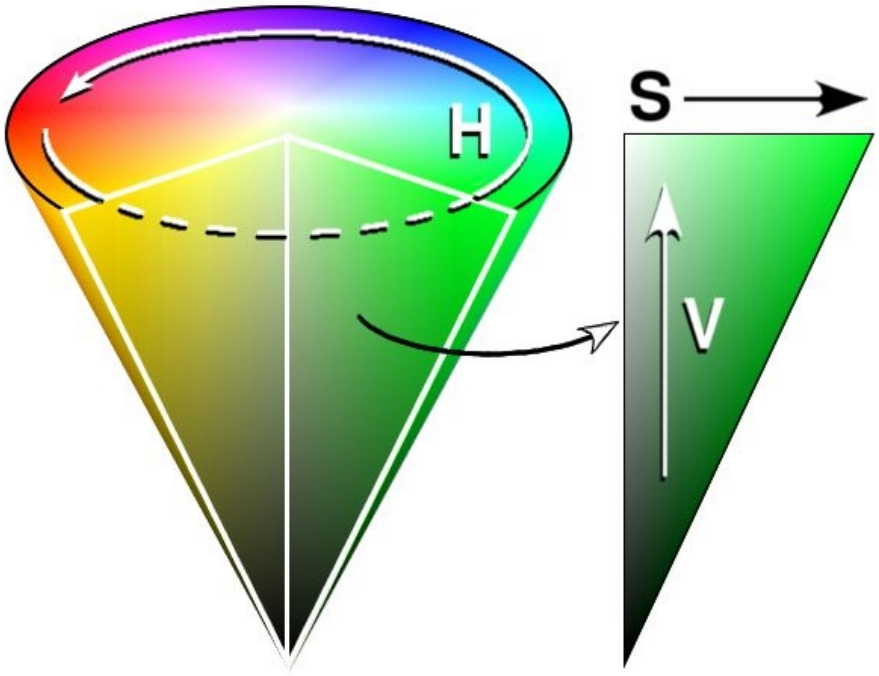

Figura 5 – Cone representativo do espaço HSV

garbage collector) o controle da memória é liberado na interface C++. Esse controle manual pode ser útil em situações nas quais se enfrenta restrições de recursos(HOWSE, 2015).

- Compatibilidade com várias plataformas que não utilizam o Java: a interface C ++ do OpenCV pode ser usada em plataformas onde o Java não está disponível ou não está instalado. Podemos reutilizar uma única base de código C ++ em Windows, Mac, Linux, iOS, WinRT e Windows Phone 8, conferindo assim flexibilidade ao programador e tornando o código em C++ reusável (HOWSE, 2015).

- Interoperabilidade com outras bibliotecas: a interface C ++ do OpenCV fornece acesso a dados de imagem como bytes brutos, que podem ser interpretados e usados por muitas outras bibliotecas C ++ ou C diretamente, sem copiar ou modificar (HOWSE, 2015).

2.9 JNI

JNI é um padrão de programação (framework) que permite a um código Java rodando em uma JVM, chamar um código C ou C++. Ele é de particular importância pois tanto o OpenCV4Android, Android SDk quanto as bibliotecas padrões do Java são todos de alguma forma dependentes dele, vale a pena ressaltar que as bibliotecas padrões

do Java são quase todas escritas sobre as bibliotecas C++ ou C. É importante entender que o JNI é uma interface de programação nativa, uma espécie de ponte entre o Java e as outras , e que neste contexto existe a necessidade de fazer a conversão entre os diferentes tipos de dados, A Figura 6 mostra os tipos primitivos em Java e seus equivalentes para a linguagem nativa.

Primitive Types and Native Equivalents

Java Type	Native Type	Description
boolean	jboolean	unsigned 8 bits
byte	jbyte	signed 8 bits
char	jchar	unsigned 16 bits
short	jshort	signed 16 bits
int	jint	signed 32 bits
long	jlong	signed 64 bits
float	jfloat	32 bits
double	jdouble	64 bits
void	void	not applicable

Figura 6 – Tabela de conversão de tipos utilizada pelo JNI

O JNI permite aos programadores escreverem métodos nativos para lidar com situações nas quais um aplicativo não pode ser escrito inteiramente na linguagem de programação Java, ou quando uma biblioteca de classes Java padrão não suporta os recursos específicos da plataforma. Também é usado para modificar um aplicativo existente (escrito em outra linguagem de programação) para ser acessível a aplicativos Java. Muitas das classes de bibliotecas padrão dependem de JNI para fornecer funcionalidades para o desenvolvedor e o usuário, e. g arquivo E / S e recursos de som. Incluir implementações de API sensíveis ao desempenho e à plataforma na biblioteca padrão permite que todas as aplicações Java acessem esta funcionalidade de forma segura e independentemente da plataforma.

A estrutura JNI permite que um método nativo use objetos Java da mesma maneira que o código Java usa esses objetos. Um método nativo pode criar objetos Java e depois inspecionar e usar esses objetos para executar suas tarefas. Um método nativo também pode inspecionar e usar objetos criados pelo código do aplicativo Java.

Um aplicativo que depende do JNI perde a portabilidade que a plataforma Java

oferece (uma solução parcial é escrever uma implementação separada do código JNI para cada plataforma deixando a cargo do Java detectar o sistema operacional e carregar a implementação correta em tempo de execução). Ao contrário do senso comum o uso de código nativo no Android geralmente não resulta em ganho de performance, mas sim, em um ganho de complexidade. No geral não se deve utilizar código nativo pura e simplesmente porque se prefere programar em C ou C++.

2.10 Melhoramento de imagem

O objetivo principal de aprimorar uma imagem é torná-la mais atraente e visualmente aceitável, geralmente são feitas modificações afim de enfatizar as bordas, reduzir o ruído. Esses tipos de operações de aprimoramento e muitos outros podem ser alcançados através da filtragem. O processo de aplicação de um filtro a uma imagem é bastante padrão independentemente do filtro utilizado. Para filtros lineares, consideramos cada pixel da imagem original, geralmente referindo-se a ele como o pixel alvo e substituindo seu valor por uma soma ponderada das intensidades associadas aos seus pixels vizinhos(BRADSKI; KAEHLER, 2008). É chamado de filtro linear porque o novo valor do pixel alvo é o resultado de uma combinação linear. Os pesos na soma ponderada são determinados por um kernel de filtro (uma máscara); isso é apenas uma sub imagem do tamanho da vizinhança que queremos considerar. A maneira de calcular o novo valor do pixel alvo é posicionando o kernel de modo que a localização do centro deste coincida com o pixel alvo e então aplica-se uma combinação linear, este processo vai se repetindo e a cada iteração temos o deslocamento do filtro, deslocamento esse que pode ser dado a passos maiores que a unidade, definindo assim uma quantidade chamada stride (passos largos em inglês) a esse processo iterativo damos o nome de convolução e o resultado deste pode ser uma imagem geralmente de tamanho menor do que a imagem original, com alguma característica realçada. Por exemplo podemos ter um filtro que detecte linhas verticas, horizontais ou inclinações arbitrarias. Podemos destacar uma técnica dentro da filtragem chamada de padding (preenchimento traduzido para o português) que consiste em preencher as bordas da imagem original com novos pixels geralmente com o valor de intensidade zero para aumentar o efeito dos pixels que se encontram nos cantos da imagem (pois estes participam de poucas combinações lineares)(MUHAMMAD, 2015). A Fig 7 Demonstra uma operação de filtragem.

2.10.1 Filtro Gaussiano

O filtro gaussiano é um operador convolucional bidimensional usado para "borrar"imagens com o intuito de remover ruídos, suavizando assim a imagem, bastante similar ao filtro médio diferenciando deste apenas pelo kernel utilizado (ou máscara) que, toma os seus valores de intensidade como sendo os valores de uma distribuição normal

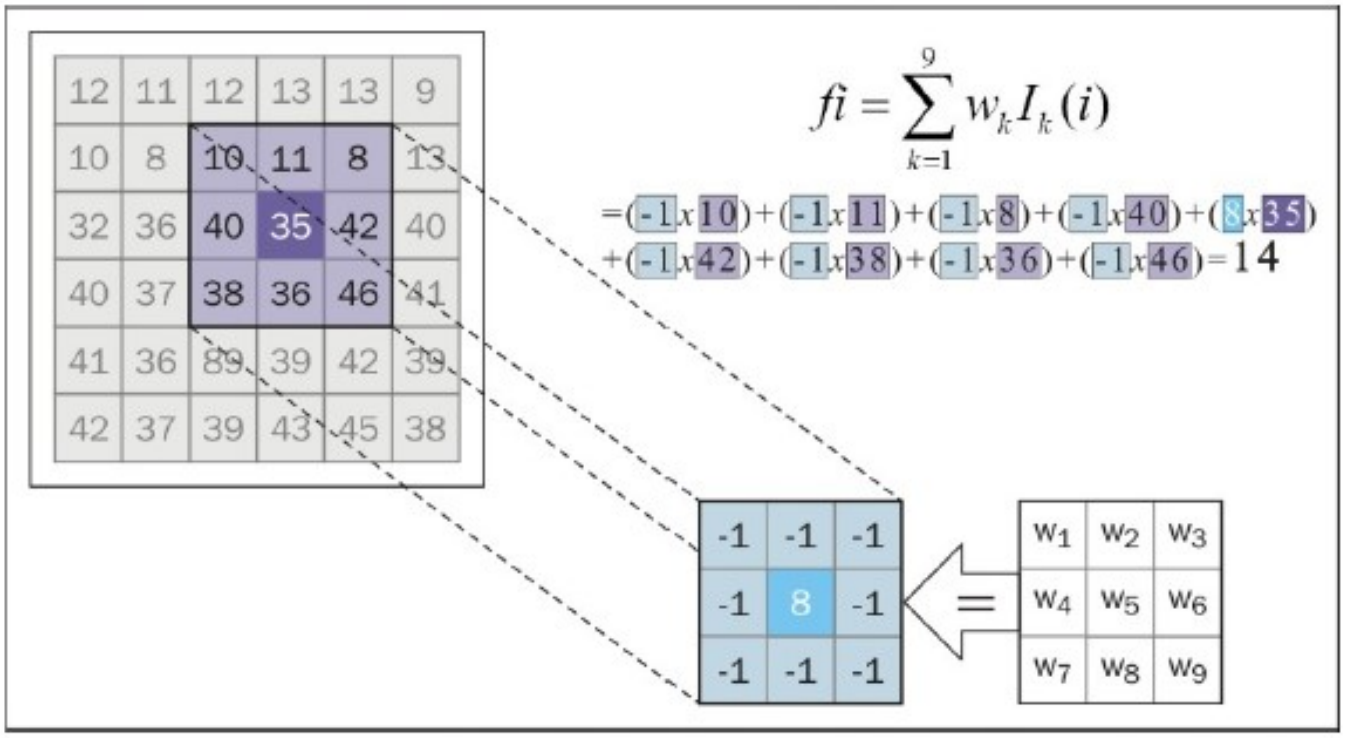

Figura 7 – Exemplo de operação de convolução a partir de um kernel

bidimensional, acentuando assim a informação do pixel central já que ele é multiplicado pelo maior valor (topo da superfície em forma de sino) e enfraquecendo a informação contida nos pixels à medida que nos afastamos do pixel central. A Figura 8. apresenta a função gaussiana utilizada para montar o kernel (BAGGIO, 2015).

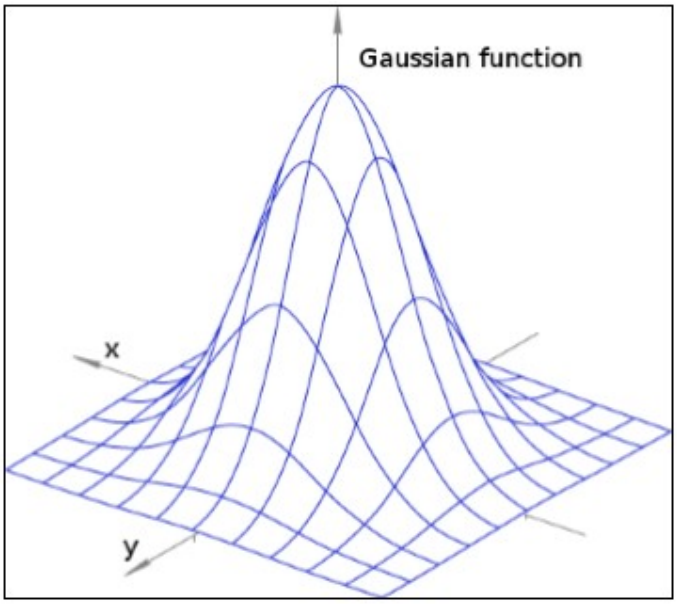

Figura 8 – Gráfico de uma função Gaussiana bidimensional

Existem alguns pontos a se considerar, uma vez que a imagem é armazenada como uma coleção de pixels discretos precisamos produzir uma aproximação discreta da distribuição gaussiana. Em teoria, a distruibuiçao normal bidimensional não é zero em toda a extensão planar, mas na prática podemos considerá-la como sendo zero após três desvios padrões a partir da média e portanto podemos destacar o kernel até este ponto a Fig. 9 mostra um kernel de convolução construído com números inteiros que é uma aproximação da função de distribuição gaussiana de desvio padrão igual à 1.0

$$\frac{1}{273}\begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix}$$

Figura 9 – Exemplo de Kernel Gaussiano

Uma vez que o kernel adequado foi calculado, a suavização gaussiana pode ser realizada usando métodos convencionais de convolução. A convolução pode, de fato, ser realizada com bastante rapidez, uma vez que a equação para o gaussiano 2-D isotrópico mostrado acima é separável nos componentes x e y. Assim, a convolução 2-D pode ser realizada primeiro convolvendo com um gaussiano na direção x e, em seguida, convolvendo com outro gaussiano na direção y. (O gaussiano é, de fato, o único operador completamente circularmente simétrico que pode ser decomposto desse jeito).

Uma maneira adicional de calcular a suavização gaussiana com um grande desvio padrão é convolver uma imagem várias vezes com um kernel gaussiano menor. Embora isso seja computacionalmente complexo, ele pode ter aplicabilidade se o processamento for realizado com computação paralela.

2.11 A Transformada Linear e Circular de Hough

A ideia original por trás da transformação linear de Hough é que qualquer ponto em uma imagem binária pode ser parte de um conjunto de possíveis linhas. Suponha que

cada linha reta possa ser parametrizada pela equação da reta $y = mx + b$, onde m é a inclinação da linha e b é a interceptação do eixo y com esta linha. Agora, poderíamos iterar toda a imagem binária, armazenando cada um dos parâmetros m e b e verificando sua acumulação. Os pontos máximos locais dos parâmetros m e b renderizariam equações de linhas retas que apareceriam na imagem para a qual o algoritmo está sendo aplicado(BRADSKI; KAEHLER, 2008). Na verdade, em vez de usar o ponto de intersecção com o eixo y (b) e a inclinação (m), utilizamos outra representação paramétrica dada pela equação $r = xcos(\theta) + ysen(\theta)$ onde os parâmetros agora são definidos como sendo: r a menor distância entre a origem e a reta e θ o angulo que esta última faz com o eixo horizontal, evitando assim o problema com as retas verticais para as quais m e b assumem valores infinitos, fato este que leva a um espaço de busca ilimitado, o que do ponto de vista computacional não é conveniente. Com esta nova representação temos que uma senóide qualquer no espaço de parametros (r, θ) é mapeada de forma unívoca a um ponto no plano (x, y) e os pontos de acumulação seriam os pontos de intersecção de todas as senóides definidas por aqueles pontos que se encontram em linha reta na imagem. Agora o problema passa a ser uma questão de se encontrar a intersecção entre essas curvas senoidais (o que computacionalmente é mais simples) e deixa de ser uma questão de descobrir se os pontos são ou não colineares, a Figura 10 representa um ponto de acumulação em lilás para o qual três pontos (três senóides no espaço paramétrico) em um plano (x, y) qualquer concorrem para ele, portanto este ponto nada mais é do que a informação de que os três pontos são colineares (estão na mesma linha reta).

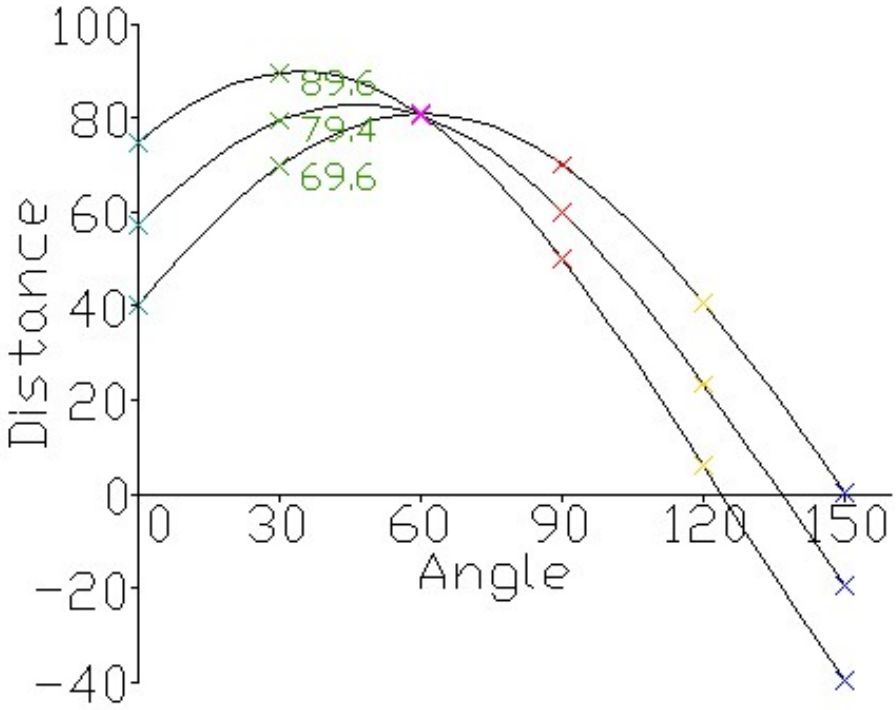

Figura 10 – Espaço paramétrico de Hough

A transformação circular é uma generalização da transformada linear de Hough. Como um círculo pode ser descrito completamente por três componentes, a saber, o centro que é composto por duas coordenadas (a, b) e o raio (R), então se um ponto (x, y) da imagem pertence ao perímetro de um círculo ele deve satisfazer as seguintes equações:

$$x = a + Rcos(\theta)$$

$$y = b + Rsin(\theta)$$

A ideia por trás é basicamente a mesma: agora cada ponto no plano representado pelas coordenadas x e y pode ser mapeado a uma superfície cônica no espaço de parâmetros de Hough, que neste caso é um espaço tridimensional visto que há três componentes: a, b e R. Agora para uma dada circunferência no plano, cada ponto (x, y) do perímetro desta, pode ser mapeado a uma circunferência de centro (x, y) e raio fixo no espaço paramétrico (a, b, R) (MUHAMMAD, 2015). A intersecção de todos os círculos mapeados em um ponto (a, b) define o centro do círculo no plano determinando pelos eixos x e y. Assim a implementação do algoritmo varre toda a imagem supondo vários círculos para cada ponto desta utilizando-se de uma matriz de acumulação para armazenar os pontos de intersecção e em seguida é calculado o máximo local que retornaria o centro do círculo encontrado. A Figura 11 ilustra uma intersecção no espaço paramétrico.

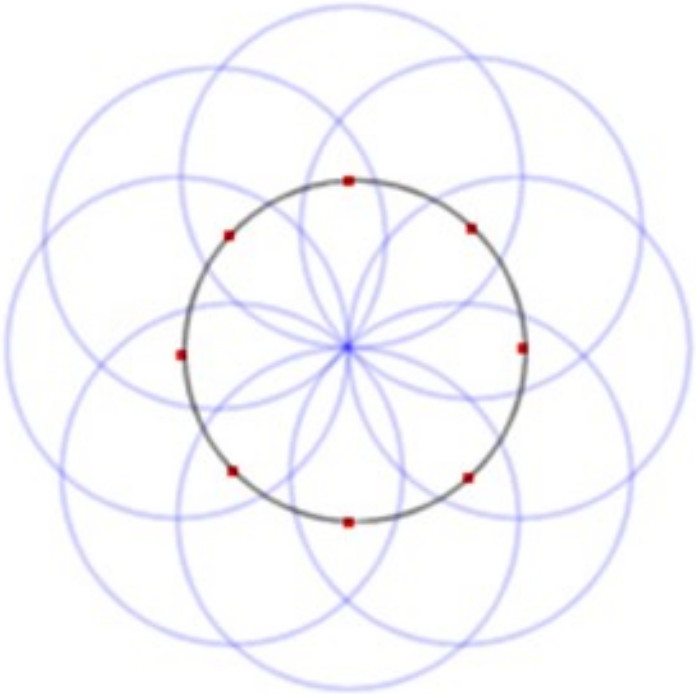

Figura 11 – Espaço paramétrico de Hough para transformada circular

2.12 O Sistema Android

Dados divulgados recentemente, pela Kantar Worldpanel mostram que o Android, sistema operacional do Google, segue na liderança do mercado mundial. Nos cinco maiores mercados do mundo, o Android segue dominando, com 76,3O sistema operacional do Android é baseado no kernel do Linux, que é responsável pelo gerenciamento da memória, processos, threads, segurança dos arquivos e pastas, além de redes e drivers (LECHETA, 2013). Cada aplicativo no Android dispara um novo processo no sistema operacional, alguns deles podem exibir uma tela para o usuário, e outros podem ficar em execução em segundo plano por tempo indeterminado. Diversos processos e aplicativos podem ser executados simultaneamente, e o kernel do sistema operacional é responsável por fazer todo esse controle de memória. Caso necessário o sistema operacional pode decidir encerrar algum processo para liberar memória e recursos, e talvez até reiniciar o mesmo processo posteriormente quando a situação voltar ao normal. Toda segurança do Android é baseada no Linux, no Android, cada aplicação é executada em um único processo e a cada processo por sua vez contem thread dedicada. Para cada aplicação instalada no celular é criado um usuário no sistema operacional para ter acesso a sua estrutura de diretórios. Dessa forma, nenhum outro usuário pode ter acesso a essa aplicação(LECHETA, 2013). A linguagem utilizada para a construção de aplicações para o Android é o Java. Como no sistema operacional não existe uma máquina virtual Java (JVM), O que se tem é uma outra máquina virtual responsável por essa tarefa, chamada de Dalvik que é otimizada para a execução em sistemas móveis. O sistema Android fornece um conjunto de recursos e funcionalidades que podem ser acessadas através de uma API Java que contém todas as bibliotecas escritas nesta linguagem (LECHETA, 2013). Para o desenvolvimento do presente trabalho foi utilizado o Android Studio, que é a IDE oficial de desenvolvimento baseada no IntelliJ IDEA da JetBrains, que acompanha SDK (em inglês, Standard Development Kit) que é um kit de desenvolvimento que contém todas ferramentas necessárias, classes auxiliares, API's (Application Programming Interface) Java, códigos de exemplo para o desenvolvimento de aplicações Android, além do NDK que também é um conjunto de ferramentas necessárias para se implementar parte do código de um aplicativo em uma outra linguagem nativa que, pode ser o C ou C++. Para este presente trabalho foi utilizado predominantemente o C++, o que ajudou a reutilizar as bibliotecas escritas nesta linguagem, já que as bibliotecas escritas em Java apresentavam alguns problemas. Dentre algumas vantagens em relação ao uso Android Studio podemos destacar:

- Editor visual mais fluido e com mais opções.
- Sistema de build mais moderno baseado em Gradle.
- Diversas utilidades e facilidades ao desenvolver para Android, sendo bastante integrado ao Android SDK.

- Templates de projetos para smartphones, tablets, relógios etc.
- Atualização e melhorias frequentes.

Uma das principais diferenças entre o Android Studio e outras IDE's é o processo de compilação dos projetos que é feita pelo Gradle, que é um moderno sistema de builds. Segundo o site oficial do Gradle ele pode ser definido da seguinte maneira: "Gradle combina o poder e a flexibilidade do Ant com o gerenciamento de dependências e convenções do Maven, em uma maneira eficaz".

3 Trabalhos Correlatos

3.1 Introdução

Este capitulo tem por objetivo apresentar a aplicação desenvolvida por (PIMENTEL et al., 2017) que serviu de base para este trabalho que tem por escopo o desenvolvimento de uma nova versão contemplando funções adicionais no tocante as informações de mapeamento dos E-Pucks. São apresentadas informações a respeito do funcionamento da aplicação além dos resultados obtidos pelo autor.

3.2 Ambiente de testes e técnicas utilizadas

Diversos projetos têm sido realizados para a construção de ambientes de testes com a finalidade de se estudar autômatos celulares para o cálculo de trajetórias. Podemos dividir esse processo em duas partes: primeiramente deve ser feito o mapeamento das disposições dos robôs, obstáculos para os quais estes possam vir a se deparar e do objetivo que deve ser alcançado por estes robôs. Feito o mapeamento entra a parte de cálculo da trajetória ótima afim de se alcançar aquele objetivo fixado no campo de testes. Sendo assim (PIMENTEL et al., 2017) desenvolveu uma aplicação para Android com foco na parte de mapeamento fornecendo assim todas as informações necessárias para que uma outra aplicação possa vir a fazer o cálculo da trajetória ótima.

(PIMENTEL et al., 2017) utilizou para realização dos experimentos, uma mesa de testes que 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. 12. Já a Figura 13 conta com um diagrama no qual são especificadas as dimensões da mesa para a qual foram realizados os testes. Como pode se notar na Figura 12 e Figura 13, uma espécie de palanque foi utilizado como suporte para sustentar o campo acima do plano do piso, este campo é constituído de madeira MDF (Medium Density Fiberboard). No lado menor do palanque foi fixado um cano de PVC (Policloreto de vinila) com uma junção em L e em seguida foi encaixado outro segmento de cano PVC, formando assim, a estrutura para alocar o smartphone ou dispositivo Android. No cano de PVC paralelo à mesa, próximo ao rasgo onde se coloca o dispositivo Android, existe um furo para passagem de um cabo USB que pode ser conectado ao dispositivo. Foi passado um cabo USB através deste furo para fazer o controle do aplicativo OCVCmakeTCCFinal, rodando no dispositivo através de um computador pessoal com conexão USB.

Como mostra a Figura 13 foram posicionados marcadores quadrados vermelhos

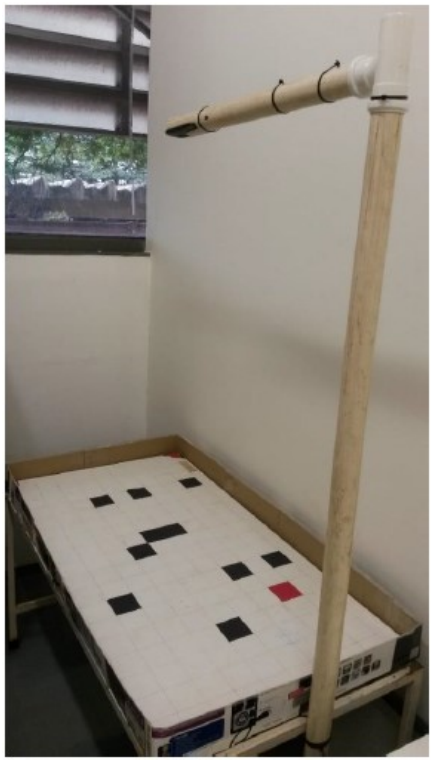

Figura 12 – Mesa de testes construída pelos integrantes do Laboratório de Computação Bio-Inspirada

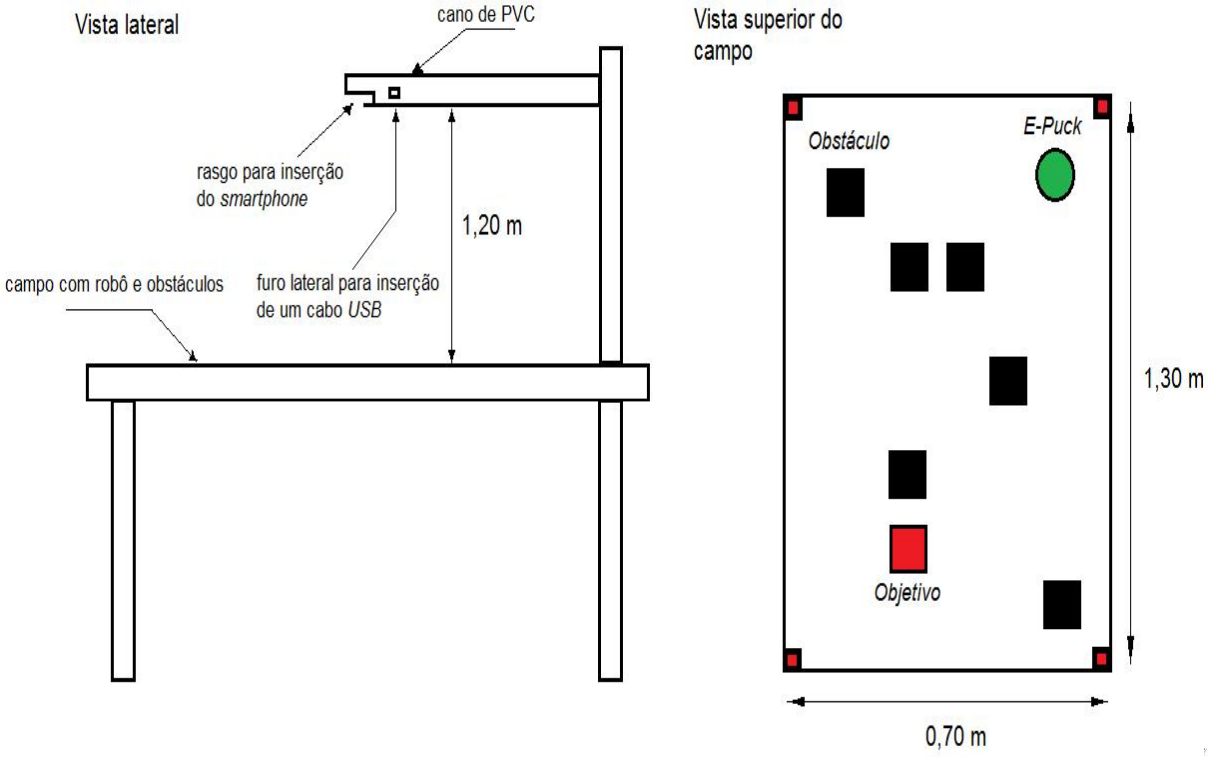

Figura 13 – Desenho esquemático com as dimensões relevantes da mesa de testes

afim de se identificar as bordas do campo, dado que a imagem abrange uma porção muito maior do que apenas o campo que é a região de interesse ROI (do inglês Region of Interest). Faz assim então necessário realizar uma filtragem para a demarcação do campo onde serão realizados os testes. Analisando ainda a Figura 13 nota se quadrados de cor preta dispostos de forma aleatória que são os obstáculos. O objetivo a ser alcançado pelo E-Puck como um quadrado de cor vermelha. O E-Puck foi representado como sendo um círculo de cor verde com o valor do diâmetro igual ao lado do quadrado definido como obstáculo.

3.3 A Aplicação Android

A interface principal da aplicação é apresentada na Fig. 14, e é composta por cinco menus principais, sendo eles: FILE, CONFIG MODE, SET FIELD, MAP e SERVER.

Figura 14 – Interface principal da aplicação, com região representativa do frame retornado pela câmera e posteriormente processado. a) com flash desligado; b) com flash ligado

O idioma padrão utilizado para a composição dos menus foi o inglês, visto que hoje em dia a utilização desse idioma é um pré-requisito fundamental para a disseminação de qualquer aplicação (PIMENTEL et al., 2017). A interface principal conta com um desenho de uma lâmpada amarela no canto superior direito, utilizado para acionar o flash da câmera (caso suportado pelo dispositivo) ou para desativa-lo (PIMENTEL et al., 2017). O flash aqui é utilizado para aquelas situações nas quais se dispões de pouca de luz, aumentando assim a iluminação ambiente e consequentemente facilitando o reconhecimento dos objetos. A Figura 14.a ilustra a interface principal com o flash desligado, e a (b) com o flash ligado.

A Fig. 14.a conta com uma região retangular em vermelho sobre a qual é feito todo o processamento dos quadros e onde é exibido ao usuário o resultado destes processamen-

tos. Essa região é denominada JavaCameraView, e o método que toma como argumento esses frames e retorna esses quadros processados é denominado onCameraFrame (PIMENTEL et al., 2017). É nesta região que o usuário poderá visualizar a imagem capturada do campo na mesa de testes e o processamento de imagem realizado sobre ela, poderá também verificar o processamento por parte dos algoritmos de visão computacional em tempo quase que real (PIMENTEL et al., 2017).Todas as configurações realizadas no menu Config podem ser visualizadas à partir deste retângulo delimitador, que para o caso especifico do trabalho realizado é uma região de 800x600 pixels. Cada dispositivo por ter uma câmera cada uma com as suas particularidades, pode ter essa região com diferentes dimensões. As imagens que são captadas pela câmera estão no formato RGBA, o que significa que, estão representadas no espaço de cor RGB e possuem um canal extra, denominado Alpha, que é responsável por controlar a transparência da imagem, porém, este canal não foi utilizado nem em Glênio (2017) nem no presente trabalho. Este retângulo vermelho apresentado na Figura 14 foi usado a exemplo de explicação já que ele é substituído pela imagem captada pela câmera.

3.3.1 Menu *FILE*

O menu FILE é responsável pelo controle e gerenciamento dos arquivos de configuração da aplicação. Todos os parâmetros utilizados em todas as opções do menu CONFIG tais como filtragem de cor do campo e objetivo da função Crop and Goal, dos obstáculos da função Obstacles, bem como os parâmetros de busca pelo E-Puck pela função Find_EPuck e pela divisão do campo em células em quantidades tanto na direção horizontal e vertical feito pela função Grid , além de todos os mapas gerados pela aplicação, podem ser salvos através do menu FILE ou utilizando-se funções empregadas pelo menu FILE (PIMENTEL et al., 2017). O menu FILE juntamente com as suas funções pode ser visualizado na Fig. 15.

O menu FILE conta com quatro funções, que podem ser vistas na Fig. 15.a. A primeira delas, a função, 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 (PIMENTEL et al., 2017).

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. 15.b. Pode se escolher um destes arquivos de nome do tipo OCVConfigData_(data)_(horário).txt onde (data) e (horário) são referentes a data e o horário respectivamente nos quais aquelas configurações foram armazenadas evitando

assim conflitos de nomes para estes arquivos, para que posteriormente possa se fazer o uso destes parâmetros dado que aquele conjunto de valores para os parâmetros se fez mais conveniente para uma determinada configuração de testes (PIMENTEL et al., 2017). O botão Cancel encerra a janela.

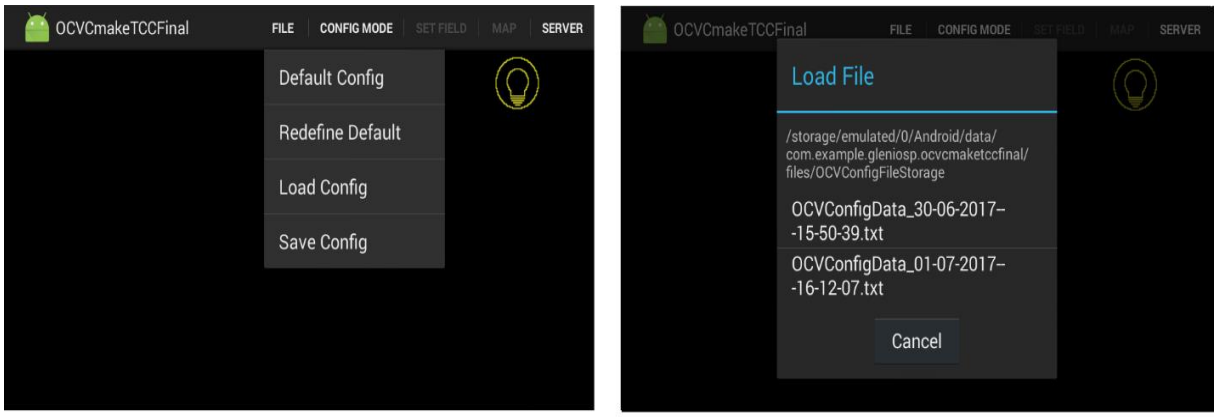

Figura 15 – Menu FILE. a) opções do menu; b) Intenção para carregamento de arquivo de configurações

O menu FILE conta também com a opção Save Config que armazena a última configuração de parâmetros estabelecida pelo usuário em um arquivo com um nome no seguinte padrão: OCVConfigData_(data)_(horário).txt, para que aquele conjunto d e valores possa ser utilizado posteriormente. Aqui ao se pressionar Save Conifg aparece sobre a tela uma Intenção que abri uma janela, com um campo de texto para que outro nome possa ser dado ao arquivo, além de fornecer um botão Cancel para fechar a janela, e um botão Save para armazenar o arquivo na memória interna do dispositivo(PIMENTEL et al., 2017).

Vale lembrar que todos os arquivos são armazenados na pasta de instalação do aplicativo na memória interna do smartphone. Caso esse armazenamento não esteja disponível ou inoperante por algum motivo, a aplicação fornece uma mensagem de aviso durante as operações tanto de armazenamento quanto de recuperação destes arquivos (PIMENTEL et al., 2017).

3.3.2 Menu *CONFIG*

Este menu é o coração da aplicação, contém todas as funções implementadas em código C++ utilizadas para o processo de mapeamento da mesa de robôs com os seus respectivos obstáculos, marcadores de campo e objetivo. Aqui pode se encontrar as configurações relativas à filtragem de cor, busca pelo E-Puck e divisão do campo em células. A Figura 15 apresenta este menu CONFIG.

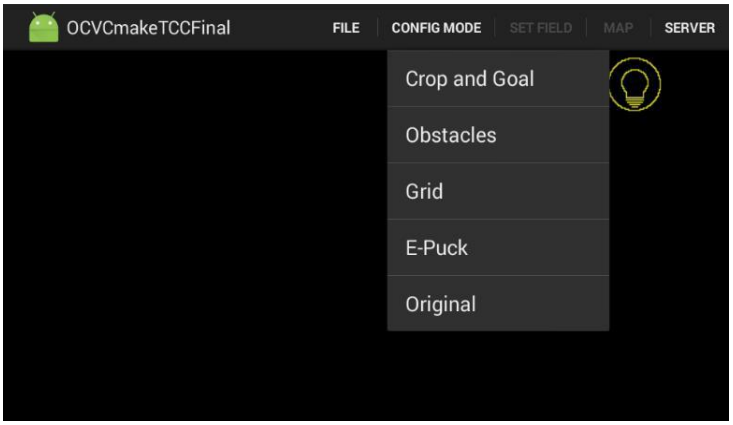

Figura 16 – Menu CONFIG da aplicação

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(PIMENTEL et al., 2017).

3.3.3 *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 (PIMENTEL et al., 2017). A cor vermelha pode ser representada por dois intervalos nas extremidades opostas dentro do conjunto de possíveis valores para o matiz. O intervalo de variação do matiz está compreendido entre 0 (inclusive) e 180 (inclusive), contrastando assim com a teoria que determina que os valores devem variar de 0 a 360 mas isto se deve apenas a uma conveniência computacional. Dito isso, o método onCameraFrame recebe o quadro capturado pela câmera e faz uma chamada à função nativa criada, goalAndFieldExtraction (PIMENTEL et al., 2017). Primeiramente este método faz a conversão dos espaços de cores de RGB para HSV a partir da função cvtColor fornecida pelo OpenCV. Assim todos os pixels que não satisfazerem os valores determinados pelos parâmetros para a cor vermelha definidos pelo usuário serão filtrados da imagem a partir de um processo de segmentação que retornará uma imagem contendo apenas aqueles objetos associados a cor vermelha, identificando assim os marcadores e o objetivo. Durante a filtragem dos pixels, operações

morfológicas de fechamento são aplicadas à imagem afim de se eliminar ruídos e pequenos contornos em vermelho esparsos assim como a aplicação de um filtro Gaussiano para suavizar a imagem.

Com os marcadores e o objetivo em vermelho destacados, forma se assim uma nova imagem g(x, y) binária associando a cada pixel em vermelho o valor 0 e ao restante o valor 1 processo este que podemos chamar de limiarização. Feito isso a função de identificação findContours da biblioteca OpenCV é utilizada para identificação dos contornos que compõe os marcadores e objetivo. Após a geração dos contornos, a função approxPolyDP, também encontrada na biblioteca OpenCV, gera a partir destes contornos o menor polígono limitante possível. Estes polígonos são então passados a função boundingRect que os utilizam formando assim o menor retângulo possível que se ajusta aquele polígono especificamente, para que enfim a partir destes possa ser feita a localização dos marcadores e objetivo no campo de testes. Estes retângulos são abstraídos pelo OpenCV pelo tipo Rect, assim os atributos de estado deste tipo tais como o ponto superior direto do retângulo e comprimento dos lados é salvo em uma variável global dentro do código fonte contendo as funções nativas C/C++ criadas (PIMENTEL et al., 2017). E em seguida, uma região de interesse que é um retângulo é desenhada com vértices nos dois centros diagonalmente opostos mais distantes dos retângulos representando os marcadores, e então esta região é salva. É dentro da região de interesse que todas as funções de mapeamento vão estar realizando seus respectivos processamentos (PIMENTEL et al., 2017).

É com este campo delimitador armazenado na memória interna definido que a aplicação com os seus outros modos de operação consegue realizar todo o mapeamento. Para o estabelecimento do campo é necessário que o menu SET FIELD seja acionado já que a aplicação necessita de um campo estático para poder realizar o seu processamento o que do ponto de vista lógico é necessário já que seria impossível realizar tal mapeamento com o campo mudando as suas dimensões a todo momento. Após selecionado SET FIELD o menu passará a indicar UNSET FIELD, e o campo se manterá fixo, podendo os marcadores serem retirados do campo, caso necessário (PIMENTEL et al., 2017). Para definir um novo campo para mapeamento, basta pressionar UNSET FIELD, e pressionar novamente SET FIELD visto que a nova configuração agrada o usuário, assim um novo campo será armazenado para o processamento dos outros modos. O modo de operação Crop and Goal com o Fragmento de configuração dos parâmetros HSV aberto o menu SET FILED pressionado (UNSET FIELD), é mostrado na Fig. 17.

3.3.4 Os outros modos de operação: *Obstacles, Grid* e *E-Puck*

O modo de operação Obstacles também conta com operações de filtro tal como o modo Crop and Goal, só que aqui o intuito é filtrar a cor preta que representa os obstáculos que os robôs possam vir a se deparar, para isto a aplicação conta com a

Figura 17 – 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

função nativa obstaclesExtraction. Novamente é utilizado todo o conjunto de funções do OPenCV tais como cvtColor para fazer a conversão do espaço de cores de RGB para HSV, findContours, approxPolyDP e boundingRect, para extrair da imagem binarizada primeiramente os contornos destes quadrados pretos depois o menor polígono que os aproxima e por último o menor retângulo possível que se ajusta aos polígonos gerados, retângulo este que fornecerá a localização do obstáculo através dos atributos do tipo Rect que é uma abstração da biblioteca OpenCV para o retângulo geométricos. Todos os parâmetros podem ser ajustados pelo usuário através de um fragmento contendo todos os valores necessários para filtrar a cor preta. A Figura 18 mostra o fragmento contendo os parâmetros ao se pressionar Obstacles.

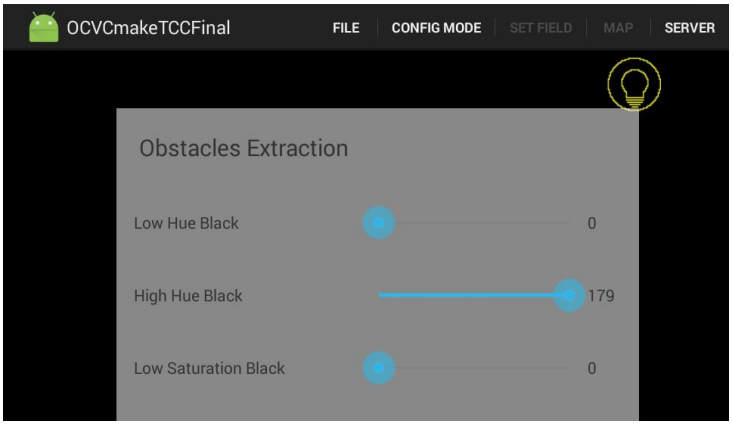

Figura 18 – Modo de operação Obstacles

O modo de operação Grid tem por função dividir o campo em células retangulares enumeradas através da chamada da função nativa divideFieldInCells. Este é o modo pelo qual o mapeamento se dá: a cada célula enumerada se atribui um estado que pode ser igual

a goal (objetivo), obstacle (obstáculo) ou epuck. Assim consegue se gerar um autômato para o cálculo de trajetória. Ao se pressionar Grid surge sobre a tela um fragmento contendo dois parâmetros que representam o número de células na horizontal e o número de células na vertical sendo representados por Number of Cells in X e Number of Cells in Y respectivamente. A Figura 19 apresenta o fragmento que surgi ao se pressionar Grid.

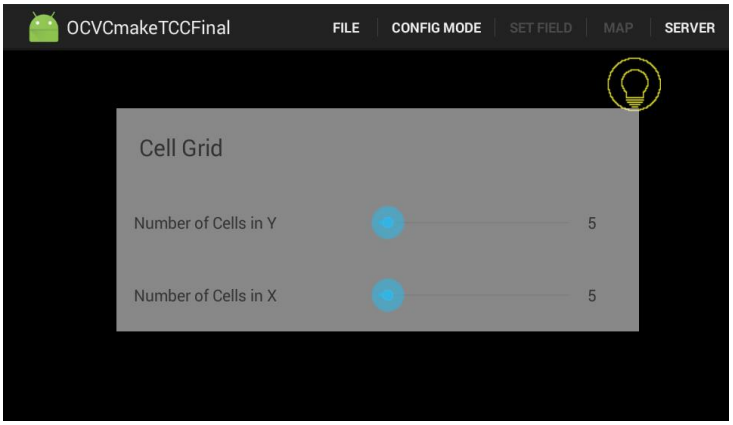

Figura 19 – Modo de configuração Grid

Já o modo de operação E-Puck realiza a identificação dos robôs E-Puck no campo de testes. Para isso tal como para os modos de operação descritos nesta seção, o campo tem que estar estabelecido através de SET FIELD. Ao se pressionar o sub-menu E-Puck um fragmento surgi na tela para configuração dos parâmetros necessários à identificação dos círculos definidos pela estrutura natural dos robôs E-Puck. A função HoughCircles da biblioteca OpenCV procura por prováveis círculos varrendo todos os pixels na imagem definida dentro da região de interesse buscando assim aqueles que possam fazer parte de um lugar geométrico circular utilizando a Transformada Circular de Hough para isto.

Neste modo de operação E-Puck é realizada a chamada à função nativa criada, findEPuck. Esta função executa primeiramente a conversão da imagem que se encontra no espaço de cores RGB para uma imagem em escala de cinza, fazendo uma chamda à função cvtColor (PIMENTEL et al., 2017). A conversão se faz necessária já que o detector de bordas de Canny utilizado pela função HoughCircles, só aceita imagens em escala de cinza como um de seus parâmetros de entrada. houghCircleTransform que é uma função nativa criada, toma essa imagem convertida em escala de cinza como um de seus parâmetros. Os parâmetros passados pelo usuário no fragmento também são passados à ela(PIMENTEL et al., 2017). Esta função faz o uso de um filtro Gaussiano sobre a imagem em escala de cinza para a suavização desta, de forma a reduzir os ruídos que possam levar o algoritmo a encontrar falsos círculos através da função HoughCircles (PIMENTEL et al., 2017). Então, com a imagem já suavizada pelo filtro Gaussiano, houghCircleTransform executa transformada de Hough através de HoughCircles com a imagem em escala de cinza e os parâmetros que foram definidos pelo usuário e passados à houghCircleTransform.

HoughCircles implementa internamente um Detector de Bordas Canny para segmentação da imagem e dá andamento com a identificação dos círculos(PIMENTEL et al., 2017). A Figura 20 apresenta o fragmento aberto ao se pressionar o modo de operação E-Puck com os seus respectivos parâmetros.

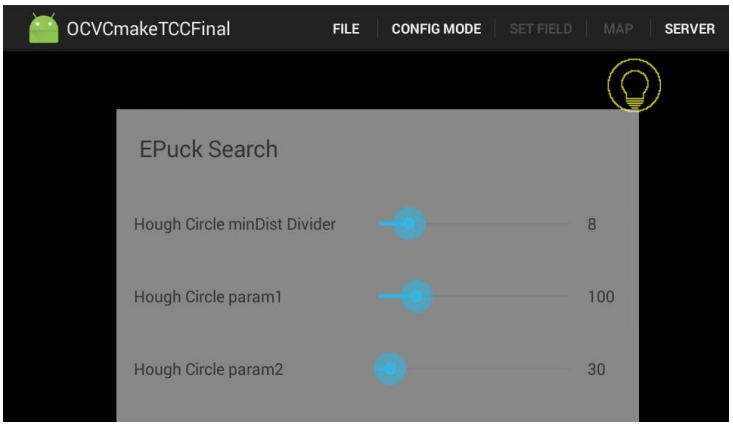

Figura 20 – Modo de operação E-Puck

3.3.5 O menu *MAP*

É neste menu que de fato se dão todas as operações de mapeamento, tendo o usuário configurado os parâmetros referentes aos modos de operação descritos anteriormente. Estas operações são disparadas pelo submenu Start Mapping que ao ser pressionado desencadeia uma série de chamadas as funções nativas tais como obstaclesExtraction, divideFieldInCells e houghCircleTransform, em uma ordem predeterminada por (PIMENTEL et al., 2017) e ao final se faz uma chamada à função nativa mapField que atribui estados a cada célula a partir das intersecções dos objetos dispostos sobre a região de interesse com as células definidas no modo de operação Grid. Assim a maior área de intersecção de um dado objeto com uma célula defini aquele objeto especificamente como sendo o estado daquela célula, por exemplo, suponha que a área de um obstáculo intersecte três células 1, 2 e 3, assim se a maior área de intersecção for com a célula de número 3 é definido como obstáculo o estado desta célula desenhando se assim um retângulo com a borda verde delimitando essa intersecção.

Esse menu possui ainda os seguintes submenus: Stop Mapping que aparece somente quando Start Mapping é pressionada dando lugar a esta que, tem a função de parar a ordem de disparo das operações efetuada por Start Mapping. Conta também com Timer Config, Timer Save e Timer Send. Timer Config através do recurso de Intenção do Android é utilizado para gerar um intervalo de tempo no qual será executado o mapeamento já que Start Mapping executa a operação de mapeamento indefinidamente, essa é uma opção conveniente e através dela é possível também estabelecer a quantidade de vezes que será

executada a operação de mapeamento. A Figura 21 apresenta uma demonstração do menu Map.

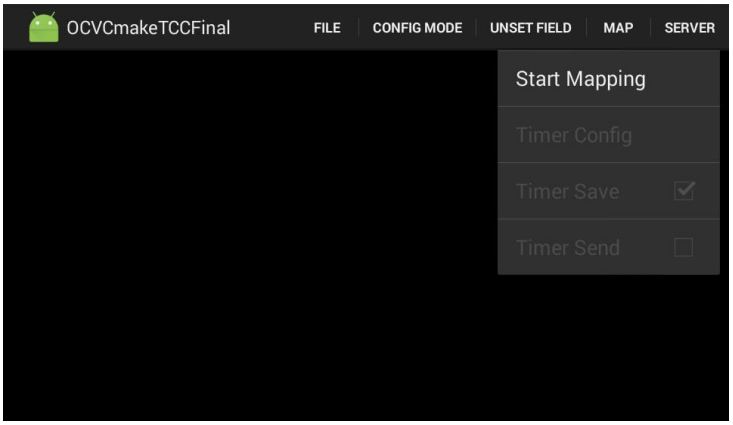

Figura 21 – Menu MAP relativo ao modo de operação Map

Todos os mapas gerados são armazenados na memória interna do dispositivo juntamente com um arquivo .txt que conta com os estados obtidos para cada célula a partir da operação de mapeamento. Eles são salvos em uma pasta chamada OCVMapFileStorage com os seguintes nomes: OCVMap_(data)_(horário).png para a figura gerada e OCVMap_(data)_(horário).txt para o arquivo de texto.

4 Metodologia

4.1 Introdução

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. Foram realizadas duas baterias de teste, onde se procurou na segunda bateria melhorar os resultados obtidos na primeira, através da remodelagem de alguns elementos. A aplicação desenvolvida pode ser encontrada no repositório do *Github* (https://github.com/RicardoCamilo/TCC).

4.2 Ambiente de teste

Para a mesa de testes na qual foram dispostos os demais elementos, foi utilizada uma cartolina de cor branca com as seguintes dimensões: 50 centímetros de largura por 66 centímetros de comprimento. Para a confecção dos obstáculos foi utilizada uma cartolina de cor preta recortada em pequenos quadrados com aproximadamente 5 centímetros de lado. Para a confecção dos marcadores e objetivo foi utilizada uma cartolina na cor vermelha recortada em quadrados com 5 e 8 centímetros de lado respectivamente. Para os robôs *E-Puck* foram feitos revestimentos em papel cartolina nas cores: amarelo, verde, azul, ciano e rosa.

A cartolina simula uma mesa de testes na qual todos os elementos foram dispostos tais como: obstáculos, marcadores para o campo , objetivo e os revestimentos cilíndricos para os robôs e na qual todo o mapeamento será realizado pela aplicação. Aqui cabe ressaltar que o fundo branco torna mais fácil a filtragem das cores. A cartolina branca foi colocada na vertical, colada junto a uma parede por fita crepe. Os marcadores foram posicionados nas bordas do campo e o objetivo colado em uma posição acima da linha média que dividi o campo. Os obstáculos foram dispostos de forma aleatória. Para a confecção dos revestimentos dos *E-Puck* para a primeira bateria de testes foram utilizadas cartolinas de cores amarela, verde, azul, ciano e rosa. Os revestimentos contam com uma forma cilíndrica respeitando a dimensão e forma do robô *E-Puck*. Aqui cada cilindro tem aproximadamente 7,75 centímetros de diâmetro por 3,5 centímetros de altura, contam ainda com um rasgo na superfície lateral em forma de retângulo afim de não comprometer a câmera alocada em cada *E-Puck*. Sobre a superfície destes revestimentos cilíndricos foram colocadas moedas de 10 centavos com aproximadamente 2 centímetro de diâmetro de forma excêntrica próximo a borda, de modo que a reta que liga o centro da moeda e o

centro do cilindro aponta na direção de deslocamento do *E-Puck*. A identificação destes centros (consequentemente da reta que indica a direção) e das cores destes revestimentos é o objetivo dos testes aqui realizados.

Para a segunda bateria de testes foram substituídos os revestimentos dos *E-Pucks* por círculos com a coloração mais forte e com aproximadamente o mesmo diâmetro, que foram confeccionados a partir de color cards e papel couchê, foi retirado a cor rosa e colocado um vermelho de coloração mais forte. A moeda de dez centavos foi substituída por círculos brancos recortados também em papel couchê de cor branca (pois estes tem mais brilho) com aproximadamente 25 milímetros de diâmetro

Como o escopo deste trabalho em suma é identificar as direções e as cores dos revestimentos dos robôs *E-Pucks*, especificando e ampliando assim as informações de mapeamento oferecidas pela aplicação anterior, foram utilizadas várias configurações de testes levando em conta a quantidade máxima estipulada no início deste projeto que era de 5 *E-Pucks*. Foi utilizado primeiramente um notebook para fixar o celular e posteriormente foi confeccionada uma estrutura em papelão, basicamente um caixa, que pudesse alocar o celular lhe conferindo firmeza e maior maleabilidade. Essa prática de fixar o dispositivo se tornou necessária já que é preferível que o celular fique estático, evitando assim possíveis perturbações na extração das imagens por parte do usuário ao manusear o dispositivo, o que dificulta o processamento dos frames pelos algoritmos internos da aplicação. Assim o controle do celular foi feito remotamente. Foram testadas diversas abordagens e a que se mostrou mais eficiente para esta tarefa foi a utilização do software AirDroid que permite o acesso e controle total da tela do celular Android, assim os ajustes para os filtros e parâmetros de busca para o *E-Puck* foram feitos a partir de um notebook via Wi-Fi sem a necessidade de cabos de conexão USB. Com as configurações iniciais realizadas, o processo de mapeamento se deu através dos seguintes passos:

- Os marcadores do campo foram identificados através da filtragem da cor vermelha e da identificação de seus contornos através da função findContours;
- Com a identificação dos marcadores, o objetivo em vermelho foi identificado, e seu contorno aproximado por um retângulo, sendo sua posição, salva na memória do dispositivo;
- Definiu-se um campo fixado na tela do dispositivo formado pelo retângulo de cor vermelha 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, as localizações dos robôs *E-Puck* foram encontradas através da Transformada Circular de Hough. No processo de localização de cada *E-Puck* é desenhado um círculo sobre o seu perímetro com aproximadamente a mesma cor do seu revestimento, também é desenhada uma seta azul apontando na direção do seu deslocamento entre os centros do revestimentos e da moeda ou círculo branco postos sobre estes. Após isto, para gravar as informações a respeito do *E-Puck* foi instanciado um retângulo (tipo Rect) que o envolvesse afim de salvar sua posição no campo estabelecido, e também um retângulo para conter as informações a respeito da sua direção (cada centro foi instanciado como sendo um ponto diagonalmente oposto deste retângulo).

- Com as posições do objetivo, dos obstáculos e dos robôs *E-Pucks*, o campo foi decomposto em células de tamanhos iguais. A quantidade de células nas direções horizontal e vertical pode ser configurada a qualquer instante na aplicação via menu;

- 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), sendo que a o estado epuck está agora acompanhado de duas novas informações: cor e um par de pontos indicando a direção deste. Todas as células que não possuíssem um destes estados, foram consideradas células livres;

5 Resultados e Discussões

5.1 Resultados

Para a primeira bateria de testes o celular foi fixado na parte posterior do notebook de forma a ganhar um grau de liberdade com o movimento da tela em torno do seu eixo próprio de rotação, fato este que ajudou bastante no ajuste da angulação e altura em relação ao campo de testes, para que a captura das imagens fossem minimamente distorcidas. Assim o notebook foi colocado sobre uma cadeira de modo que a altura da câmera do smartphone ficasse um nível a cima do centroide do campo de testes, evitando assim mais uma vez, a captura de imagens distorcidas. A cartolina de cor branca foi pregada junto a parede com fita crepe e foram dispostos os marcadores do campo, quatro no total, sobre os cantos da cartolina. Foram distribuídos quatro obstáculos sobre a superfície desta, de modo aleatório e um objetivo na parte superior da cartolina. De início o objetivo tinha as mesmas dimensões dos marcadores mas, ao se realizar os primeiros testes, notou se uma dificuldade para identifica-lo no campo, fato que acarretou na decisão de aumentar suas dimensões. O celular foi plugado a entrada USB do notebook e foi realizado o controle remoto deste a partir do software *Airdroid*. Foi utilizado também o aplicativo *Recordable* para gravar as sessões de teste.

Foram estabelecidas três configurações de teste:

- 25 células, 5 na horizontal e 5 na vertical. foram distribuídos 5 revestimentos para *E-Puck* de cores: amarelo, azul escuro, ciano, verde e rosa.
- 25 células, 5 na horizontal e 5 na vertical. foram distribuídos 4 revestimentos para *E-Puck* de cores: amarelo, azul escuro, ciano e verde. Retirou se o revestimento de cor rosa.
- 25 células, 5 na horizontal e 5 na vertical. foram distribuídos 4 revestimentos para *E-Puck* de cores: amarelo, azul escuro, ciano e verde. Agora variando se a posição do revestimento de cor amarela.

Para cada configuração foi utilizado um timer de dez segundos e um intervalo de coleta de dois segundos, totalizando assim 5 coletas. As tabelas apresentam os resultados obtidos:

Tabela 1 – Primeira Configuração

	Média de acertos (Para as 5 coletas)	Cor	Direção
Objetivo	100%	Não se aplica	Não se aplica
Obstáculos	15%	Não se aplica	Não se aplica
E-Puck "Amarelo"	100%	100%	100%
E-Puck "Verde"	20%	100%	100%
E-Puck "Azul Escuro"	60%	100%	100%
E-Puck "Ciano"	20%	100%	100%
E-Puck "Vermelho"	80%	0%	100%

Tabela 2 – Segunda Configuração

	Média de acertos (Para as 5 coletas)	Cor	Direção
Objetivo	100%	Não se aplica	Não se aplica
Obstáculos	100%	Não se aplica	Não se aplica
E-Puck "Amarelo"	100%	100%	100%
E-Puck "Verde"	80%	100%	100%
E-Puck "Azul Escuro"	100%	100%	100%
E-Puck "Ciano"	20%	100%	100%
E-Puck "Vermelho"			

Tabela 3 – Terceira Configuração

	Média de acertos (Para as 5 coletas)	Cor	Direção
Objetivo	100%	Não se aplica	Não se aplica
Obstáculos	100%	Não se aplica	Não se aplica
E-Puck "Amarelo"	100%	100%	100%
E-Puck "Verde"	80%	100%	100%
E-Puck "Azul Escuro"	100%	100%	100%
E-Puck "Ciano"	20%	100%	100%
E-Puck "Vermelho"			

Para a segunda bateria de testes foi utilizado outro dispositivo, um celular *Moto G* segunda geração com as seguintes especificações:

- Arquitetura - Chipset: 32bits - Qualcomm Snapdragon 400 MSM8226

- CPU Processador / Núcleos: 1.2Ghz Quad-Core ARM Cortex-A7
- GPU Placa Gráfica: Qualcomm Adreno 305
- Memória RAM: 1GB LPDDR2
- Memória Interna: 16GB (13GB acessível pelo usuário)
- Tipo da tela: TFT LCD IPS
- Tamanho em polegadas: 5"polegadas
- Resolução da tela: 720 x 1280 pixels
- Densidade (pixels por polegadas): 294 PPI
- Câmera principal: 8 megapixels
- Resolução câmera principal: 3264 x 2448 pixels
- Gravação de vídeo câmera principal: HD (1280 x 720 pixels) 30 fps
- Flash: Flash LED

Figura 22 – Motorola Moto G 2014 XT1069 Dual DTV

O celular foi fixado dentro de uma estrutura feita a partir de uma caixa de papelão onde, foram retiradas a tampa e uma das superfícies laterais e feito dois rasgos: um para

a passagem de um cabo *USB* e outro para que a câmera do celular pudesse ficar exposta. Isso permitiu que o celular fosse manuseado durante os testes caso necessário. A estrutura foi colocada a uma altura um pouco acima do centróide do campo de testes, sobre uma cadeira. A distância entre o campo e o dispositivo teve de ser ajustada para realizar as gravações dos testes, já que era necessário por vezes, que o usuário entrasse na cena para movimentar algum *E-Puck* (representado por círculos confeccionados em color cards) fato este que poderia travar a aplicação dependendo da distância mencionada. A cartolina de cor branca foi pregada junto a parede com fita crepe e foram dispostos os marcadores do campo, quatro no total, sobre os cantos da cartolina. Foram distribuídos seis obstáculos desta vez sobre a superfície de modo aleatório e um objetivo um pouco maior do que os marcadores do campo em vermelho, na parte superior da cartolina. O celular foi plugado a entrada USB do notebook e foi realizado o controle remoto deste a partir do software *Airdroid*. Aqui para gravação dos testes foi utilizado outro aplicativo cujo nome é *DU Recorder*.As Figuras 23, 24 e 25 apresentam os quadros coletados durante o processo de mapeamento para todas as configurações de testes estabelecidas.

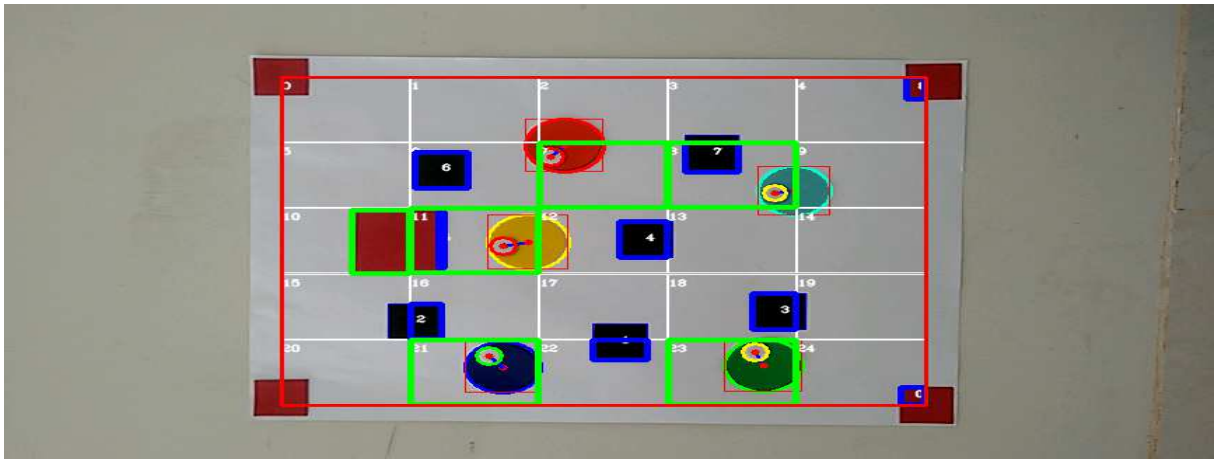

Figura 23 – Quadro coletado durante o processo de mapeamento para a primeira configuração

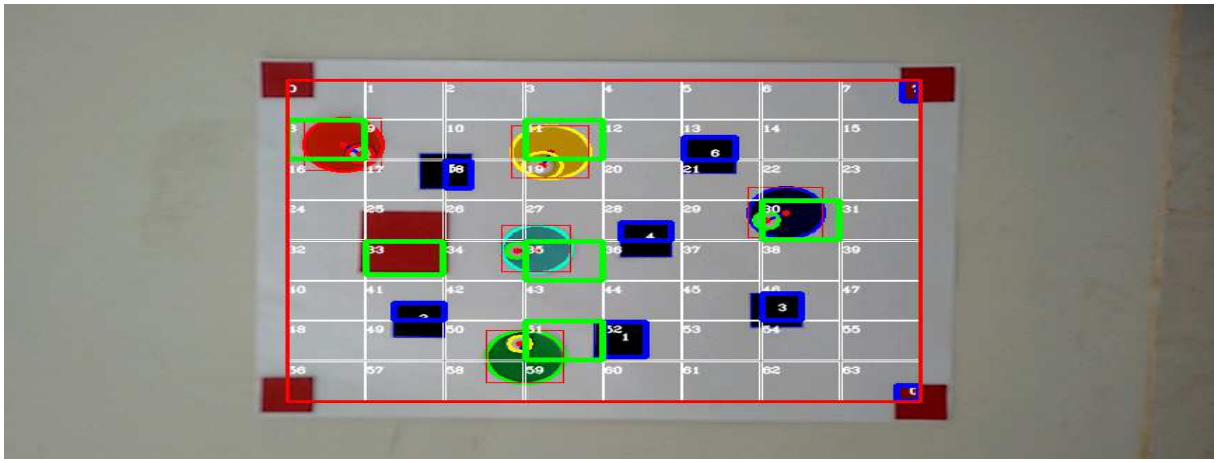

Figura 24 – Quadro coletado durante o processo de mapeamento para a segunda configuração

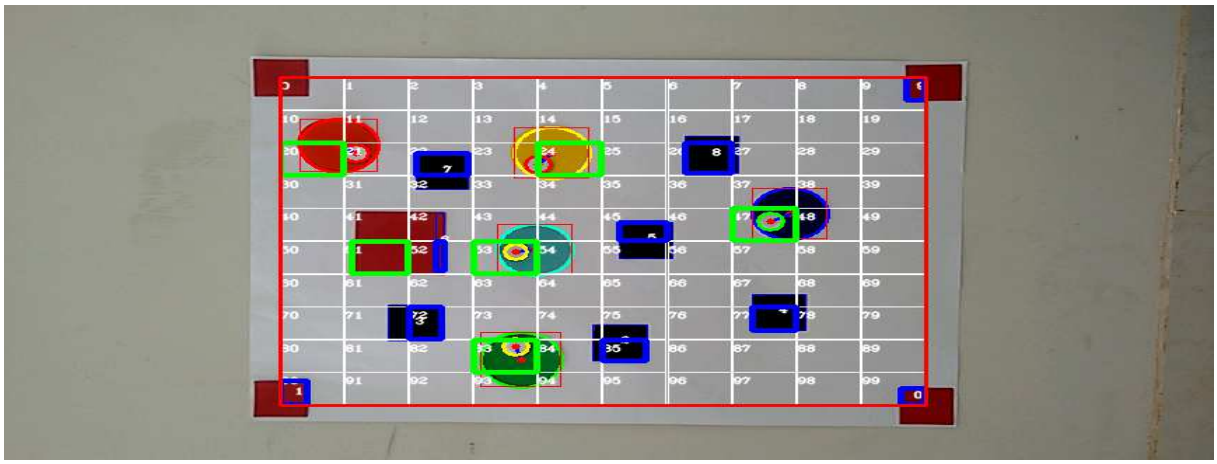

Figura 25 – Quadro coletado durante o processo de mapeamento para a terceira configuração

Na aplicação desenvolvida por (PIMENTEL et al., 2017) os retângulos com bordas verdes encerravam apenas as intersecções de maior área entre o *E-Puck* e uma célula qualquer do grid, sobre a qual ele estava alocado (este retângulo na verdade era uma aproximação desta área). Este recurso foi utilizado para fins de representação para que os usuários da aplicação pudessem verificar em qual célula do grid o robô *E-Puck* foi mapeado. Para aplicação desenvolvida por este trabalho temos que agora o retângulo encerra a célula como um todo, utilizando novamente o critério da intersecção com maior área possível.

- 25 células, 5 na horizontal e 5 na vertical. foram distribuídos 5 círculos representando os *E-Puck's* nas cores: amarelo, azul escuro, ciano, verde e vermelho.

- 64 células, 8 na horizontal e 8 na vertical. foram distribuídos 5 círculos representando os *E-Puck's* nas cores: amarelo, azul escuro, ciano, verde e vermelho
- 100 células, 10 na horizontal e 10 na vertical. foram distribuídos 5 círculos representando os *E-Puck's* nas cores: amarelo, azul escuro, ciano, verde e vermelho

Novamente para cada configuração dessa bateria de testes foi utilizado um timer de dez segundos e um intervalo de coleta de dois segundos, totalizando assim 5 coletas. As tabelas a seguir apresentam os resultados obtidos:

Tabela 4 – Tabela 4. Resultados obtidos para a primeira configuração segunda bateria do experimento: 25 células

	Média de acertos (Para as 5 coletas)	Cor	Direção
Objetivo	100%	Não se aplica	Não se aplica
Obstáculos	100%	Não se aplica	Não se aplica
E-Puck "Amarelo"	40%	100%	100%
E-Puck "Verde"	100%	80%	100%
E-Puck "Azul Escuro"	100%	100%	100%
E-Puck "Ciano"	80%	80%	100%
E-Puck "Vermelho"	60%	80%	100%

Tabela 5 – Tabela 5. Resultados obtidos para a segunda configuração segunda bateria do experimento: 64 células

	Média de acertos (Para as 5 coletas)	Cor	Direção
Objetivo	100%	Não se aplica	Não se aplica
Obstáculos	100%	Não se aplica	Não se aplica
E-Puck "Amarelo"	80%	100%	100%
E-Puck "Verde"	100%	80%	100%
E-Puck "Azul Escuro"	100%	100%	100%
E-Puck "Ciano"	100%	100%	100%
E-Puck "Vermelho"	100%	100%	100%

Tabela 6 – Tabela 6. Resultados obtidos para a terceira configuração segunda bateria do experimento: 100 células

	Média de acertos (Para as 5 coletas)	Cor	Direção
Objetivo	100%	Não se aplica	Não se aplica
Obstáculos	100%	Não se aplica	Não se aplica
E-Puck "Amarelo"	80%	100%	100%
E-Puck "Verde"	100%	80%	100%
E-Puck "Azul Escuro"	100%	100%	100%
E-Puck "Ciano"	100%	80%	100%
E-Puck "Vermelho"	100%	60%	100%

5.2 Discussão

Para a primeira bateria de testes, constatou se:

Para a primeira configuração que era um arranjo de cinco revestimentos, notou se uma boa acurácia tanto na identificação do objeto em si, quanto na direção (toda vez que um círculo foi identificado a sua direção também o foi), já para as cores, o algoritmo demonstrou facilidade em reconhecer a cor amarela, verde e azul ciano, sendo a primeira com 100% de acurácia. A cor vermelha sempre foi reconhecida como sendo um azul de matiz mais alto, o que de certa forma não é ruim, pois o matiz azul (contando com o lilás) esta bem próximo do intervalo superior do vermelho no espaço *HSV*. Outro motivo pode ter sido o fato que, o valor do matiz retornado pelo algoritmo, ser nada mais nada menos do que a média dos valores *Hue* para uma janela retangular que abriga o círculo, como nem todos os pixels que foram iterados sobre esta janela tem o mesmo valor de matiz do revestimento, haverá distorções no resultado obtido. Já o número de obstáculos identificados foi muito baixo, problema este que pode ter sido devido as dimensões da mesa ou pela quantidade de revestimentos colocados sobre o mapa. Assim sendo, para verificar se existia correlação entre o número de revestimentos dispostos e a capacidade do algoritmo de reconhecer os obstáculos, foi retirado um dos revestimentos, o de cor vermelha e realizado novamente os testes com o campo dividido no mesmo número de células, 25. Cabe ressaltar que houveram tentativas de se dividir o campo em um número maior de células, fato este que levou aplicação a um erro e seu posterior fechamento. Não houve problema com a identificação do objetivo que alcançou 100% de acurácia no processo de identificação.

Os resultados para a segunda configuração foram melhores em relação ao primeiro, aqui o algoritmo mapeou com 100% de acurácia todos os obstáculos e manteve os resultados para o revestimento de cor amarela. Houve uma melhora substancial em relação ao

revestimento de cor verde que alcançou 80% de acurácia sendo a cor verde identificada 100% das vezes. A cor azul novamente foi identificada como azul ciano, o que para este trabalho foi considerado um acerto já que este resultado é uma média (os valores do matiz são calculados com base em uma iteração feita sobre a janela definida pelo círculo encontrado na transformada de Hough) e está bem próximo do azul de coloração mais escura no espaço *HSV*. O revestimento ciano claro, foi mapeado 20% das vezes, porém a cor foi identificada de maneira correta. Todas as vezes que um revestimento foi identificado a sua direção também o foi caracterizando assim uma taxa de acertos de 100%. O objetivo foi identificado 100% das vezes. Os resultados da segunda configuração se mantiveram em relação aos da segunda configuração.

Para segunda bateria de testes constatou-se:

Com a mudança dos elementos que compunham o campo a aplicação respondeu de forma bem melhor, tanto no mapeamento quanto no reconhecimento em tempo real. Fazendo uma análise em tempo real a aplicação conseguiu praticamente 100% do reconhecimento dos elementos que representavam os robôs, com alguns erros para o acerto das cores. Outra melhora foi o fato de que a aplicação reconheceu os cinco elementos modelados para os robôs sem que para isso fosse perdida eficácia no processo de reconhecimento dos obstáculos.

Para a primeira configuração com 25 células no total houve problemas para se mapear a cor amarela e vermelha (que seriam cores próximas em relação a coordenada angular do espaço *HSV* que representa o matiz), mas suas cores foram bem identificadas dentro do intervalo de 80 a 100 por cento. o restante das cores se saíram muito bem, a direção repetiu novamente o sucesso da primeira bateria e foi identificada 100% das vezes em relação às vezes em que o elemento foi mapeado. A segunda configuração alcançou os melhores resultados isso pode ser devido a dois fatores: o número de células em que o campo foi dividido, que aparentemente não apresenta correlação dedutível e a iluminação ambiente sendo este o mais provável de ter impactado nos resultados de forma positiva. A terceira configuração teve resultados um pouco pior em relação a segunda corroborando o fato de que o número de células no grid não influência positivamente o resultado.

6 Conclusão

6.1 Conclusão

Para que aplicações de visão computacional sejam bem sucedidas é necessário que a tríade hardware, software e meio estejam em equilíbrio, sendo que a escolha deste último necessita de uma atenção especial já que as inúmeras possibilidades de modelagem tornam o processo um esforço de tentativa e erro. Surgem assim diversas dificuldades, dentre elas, controlar a relação entre luz ambiente e as cores utilizadas para os revestimentos. Neste sentido a utilização tanto de algoritmos de segmentação quanto da transformada de Hough se mostraram ferramentas poderosas para extração das informações da mesa de testes as quais esse trabalho se propôs a coletar, mostrando robustez na hora de lidar com essas dificuldades. Além disso esses algoritmos fornecem soluções de baixo custo para estudos mais aprofundados, que possam por exemplo utilizar o dados adquiridos para o treinamento de robôs.

A aplicação *Android* demonstrou assim mais uma vez seu caráter modular, permitindo a inserção de novas funcionalidades sem que que para isso fosse necessário comprometer o restante da aplicação. Em ambientes de teste onde a maior da parte das variáveis podem ser controladas ela demonstrou um alto grau de confiança, dado que em grande parte das coletas ela reconheceu mais de 80% de todas as cores e atingiu 100% no tocante ao reconhecimento das direções de deslocamento dos revestimentos.

Referências

BAGGIO, D. L. *OpenCV 3.0 Computer Vision with Java*. [S.l.]: Packt Publishing Ltd, 2015. Citado 2 vezes nas páginas 19 e 23.

BRADSKI, G.; KAEHLER, A. *Learning OpenCV: Computer vision with the OpenCV library*. [S.l.]: "O'Reilly Media, Inc.", 2008. Citado 2 vezes nas páginas 22 e 25.

CONTRIBUTORS, W. *HSL and HSV — Wikipedia, The Free Encyclopedia*. 2017. [Online; accessed 19-December-2017]. Disponível em: <https://en.wikipedia.org/w/index.php?title=HSL_and_HSV&oldid=815177738>. Citado na página 19.

GARCÍA, G. B. et al. *Learning Image Processing with OpenCV*. [S.l.]: Packt Publishing Ltd, 2015. Citado na página 14.

GONZALEZ, R. C.; WOODS, R. E. *Processamento de imagens digitais*. [S.l.]: Edgard Blucher, 2000. Citado 2 vezes nas páginas 16 e 17.

HOWSE, J. *Android application programming with OpenCV 3*. [S.l.]: Packt Publishing Ltd, 2015. Citado na página 20.

LECHETA, R. R. *Google Android-3ª Edição: Aprenda a criar aplicações para dispositivos móveis com o Android SDK*. [S.l.]: Novatec Editora, 2013. Citado na página 27.

MUHAMMAD, A. *OpenCV Android Programming By Example*. [S.l.]: Packt Publishing Ltd, 2015. Citado 2 vezes nas páginas 22 e 26.

PIMENTEL, G. S. et al. Desenvolvimento de uma aplicação android para mapeamento de uma mesa com obstáculos e robôs e-puck utilizando visão computacional. Universidade Federal de Uberlândia, 2017. Citado 12 vezes nas páginas 4, 11, 13, 29, 31, 32, 33, 34, 35, 37, 38 e 47.

QUEIROZ, J. E. R. de; GOMES, H. M. Introdução ao processamento digital de imagens. *RITA*, v. 13, n. 2, p. 11–42, 2006. Citado 2 vezes nas páginas 15 e 18.

RESNICK, R.; HALLIDAY, D.; WALKER, J. *Fundamentos de física*. [S.l.]: Continental, 1987. Citado na página 15.

SZELISKI, R. *Computer vision: algorithms and applications*. [S.l.]: Springer Science & Business Media, 2010. Citado na página 14.

WIKIPéDIA. *Visão computacional — Wikipédia, a enciclopédia livre*. 2017. [Online; accessed 15-julho-2017]. Disponível em: <https://pt.wikipedia.org/w/index.php?title=Vis%C3%A3o_computacional&oldid=49301451>. Citado na página 15.