

UNIVERSIDADE ESTADUAL PAULISTA “JÚLIO DE MESQUITA FILHO”
FACULDADE DE CIÊNCIAS

CARLOS ANDRÉ BRAILE PRZEWODOWSKI FILHO

**ROBÔ MÓVEL PARA EXPLORAÇÃO DE AMBIENTES EM BUSCA DE OBJETOS POR
MEIO DE VISÃO COMPUTACIONAL**

BAURU, SP

2015

CARLOS ANDRÉ BRAILE PRZEWODOWSKI FILHO

**ROBÔ MÓVEL PARA EXPLORAÇÃO DE AMBIENTES EM BUSCA DE OBJETOS POR
MEIO DE VISÃO COMPUTACIONAL**

Trabalho de Conclusão de Curso de graduação apresentado à disciplina “Projeto e Implementação de Sistemas” do curso de Bacharelado em Ciência da Computação da Faculdade de Ciências da Universidade Estadual Paulista “Júlio de Mesquita Filho”, como requisito parcial para obtenção do título de bacharel.

Orientador: Humberto Ferasoli Filho

BAURU, SP

2015

CARLOS ANDRÉ BRAILE PRZEWODOWSKI FILHO

**ROBÔ MÓVEL PARA EXPLORAÇÃO DE AMBIENTES EM BUSCA DE OBJETOS POR
MEIO DE VISÃO COMPUTACIONAL**

Trabalho de Conclusão de Curso de graduação apresentado à disciplina “Projeto e Implementação de Sistemas” do curso de Bacharelado em Ciência da Computação da Faculdade de Ciências da Universidade Estadual Paulista “Júlio de Mesquita Filho”, como requisito parcial para obtenção do título de bacharel.

Aprovado em ____/____/____

BANCA EXAMINADORA

Profa. Dra. Andréa Carla Gonçalves Vianna

Prof. Dr. Humberto Ferasoli Filho

Prof. Dr. Aparecido Nilceu Marana

Agradecimentos

Agradeço, primeiramente, ao meu irmão Henrique Ribeiro por ter me dado a ideia deste trabalho e por sempre ser meu grande companheiro.

Ao professor Humberto Ferasoli pela dedicação e muita paciência ao me orientar e por ser, acima de tudo, um amigo desde o começo da minha graduação.

Aos meus pais, que sempre me apoiaram e se esforçaram muito para me dar uma boa educação.

À minha família, que sempre me incentivou a estudar e sempre me acolheu quando eu precisei.

Aos meus amigos, os quais estiveram comigo em momentos de felicidade e de tristeza.

Ao Silas, ao Takao e aos funcionários do Laboratório de Robótica por terem me auxiliado diversas vezes em meus projetos e que fizeram dos meus dias no laboratório muito menos estressantes.

“Não sabendo que era impossível, foi lá e fez.”

(Jean Cocteau).

RESUMO

A robótica móvel está cada vez mais presente no nosso cotidiano e os robôs estão sendo utilizados em várias aplicações. Trabalhar em conjunto com humanos, cooperando e/ou concorrendo, é o desafio e os próximos passos da robótica móvel. O projeto apresenta o planejamento, construção e implementação de uma arquitetura de controle de *hardware* e de *software* para um robô de serviço móvel, eficiente e versátil, para realização de tarefas com o uso de técnicas de visão computacional exemplificado numa aplicação de coletar bolas de tênis em um ambiente. O trabalho apresenta desenvolvimento da base móvel, da eletrônica embarcada, do sistema de comunicação, do sistema de controle e do sistema de visão computacional, baseada na análise de cores. O robô móvel, projetado e implementado, foi chamado de *Meg-Alie* e, considerando os resultados obtidos nos experimentos realizados, ele atingiu o seu objetivo proposto.

Palavras-chave: robótica móvel, visão computacional, dispositivos móveis.

ABSTRACT

Mobile robotics is increasingly present in our daily life and robots are being used in various applications. Working together with humans, cooperating and / or running, is the challenge and the next steps of mobile robotics. The project presents the planning, construction and implementation of hardware and software control architecture for a mobile service robot, efficient and versatile, for performing tasks using computer vision techniques exemplified in an application to collect tennis balls in an environment. The project presents the development of the mobile base of embedded electronics, communication system, control system and computer vision system based on color analysis. The mobile robot, designed and implemented, was called *Meg-Alie* and, considering the results obtained in the experiments, it reached its objective.

Keywords: mobile robotics, computer vision, mobile devices.

LISTA DE FIGURAS

FIGURA 1 – ROBURGUER E SEU DISPOSITIVO MÓVEL	12
FIGURA 2 – SENSOR DE DISTÂNCIA INFRAVERMELHO	17
FIGURA 3 – SENSOR DE DISTÂNCIA ULTRASSÔNICO HC-SR04	17
FIGURA 4 – FUNCIONAMENTO DO SERVO MOTOR	19
FIGURA 5 – DIFERENTES MODELOS DE ARDUINO	20
FIGURA 6 – IDE ARDUINO	21
FIGURA 7 – CICLO DE VIDA DE UMA APLICAÇÃO ANDROID	22
FIGURA 8 – CORES ADITIVAS	26
FIGURA 9 – HSV CILÍNDRICO	27
FIGURA 10 – EXEMPLO DE DISTRIBUIÇÃO DE CORES DE UMA IMAGEM	27
FIGURA 11 – IMAGEM ORIGINAL (ESQUERDA) E EFEITO DO OPERADOR DE SOBEL APLICADO NA DIREÇÃO Y (DIREITA)	28
FIGURA 12 – IMAGEM ORIGINAL (ESQUERDA) E DETECCÃO DE BORDAS DE CANNY (DIREITA)	29
FIGURA 13 – ARQUITETURAS DE CONTROLE	30
FIGURA 14 – REPRESENTAÇÃO DO ROBÔ NO PLANO CARTESIANO	32
FIGURA 15 – EXEMPLO DE <i>OPEN LOOP</i>	33
FIGURA 16 – CAMPOS POTENCIAIS	34
FIGURA 17 – COMPARAÇÃO ENTRE OS QUADROS DA IMAGEM	35
FIGURA 18 – <i>MEG-ALIE</i>	36
FIGURA 19 – CICLO DA CAMADA INFERIOR	37
FIGURA 20 – ESTRUTURA DA APLICAÇÃO ANDROID	39
FIGURA 21 – INTERFACE DA APLICAÇÃO	39
FIGURA 22 – FLUXO DO PROCESSO DE BUSCA E COLETA DE OBJETOS	40
FIGURA 23 – AMBIENTE DE ATUAÇÃO DO SISTEMA	42

LISTA DE TABELAS

TABELA 1 - COMPONENTES BÁSICOS PARA PROGRAMAÇÃO ANDROID	22
TABELA 2 - ALGUNS SENSORES DISPONÍVEIS PARA ANDROID	24
TABELA 3 – INSTRUÇÕES DA CAMADA INFERIOR	38

SUMÁRIO

1. INTRODUÇÃO	11
1.1 OBJETIVO	13
1.2 PROPOSTA	13
1.3 MATERIAL E METODOLOGIA	13
1.3.1 Material	13
1.3.2 Metodologia	14
1.4 ORGANIZAÇÃO DA MONOGRAFIA	14
2. SISTEMAS EMBARCADOS	16
2.1 ELETRÔNICA EMBARCADA	16
2.1.1 Sensores	16
2.1.2 Atuadores	17
2.1.3 O microcontrolador embarcado - <i>Arduino</i>	19
2.2 O SMARTPHONE	21
2.2.1 Acesso à câmera do smartphone	23
2.2.2 Acesso aos sensores do smartphone	23
3 COMUNICAÇÃO ENTRE OS SISTEMAS	25
4 VISÃO COMPUTACIONAL	26
4.1 REPRESENTAÇÃO DE CORES	26
4.2 HISTOGRAMAS	27
4.3 DETECÇÃO DE BORDAS	28
5. ROBÓTICA MÓVEL AUTÔNOMA	30
5.1 ARQUITETURA DE CONTROLE	30
5.2 TRAÇÃO DIFERENCIAL	31
5.3 TÉCNICAS DE NAVEGAÇÃO	32
6. DESENVOLVIMENTO DAS CAMADAS DO ROBÔ PROPOSTO	36
6.1 CAMADA INFERIOR	36
6.2 CAMADA INTERMEDIÁRIA	38

6.3 CAMADA SUPERIOR	41
7. TESTES E RESULTADOS	42
8. CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS	44
REFERÊNCIAS	45

1. INTRODUÇÃO

A robótica tem se mostrado cada vez mais presente no cotidiano. Os robôs tem ajudado o ser-humano a realizar tarefas complexas, repetitivas ou ainda exaustivas, tais como a de exploração de ambientes hostis, produção em linhas de montagem, coleta de materiais entre tantas outras. Existem diversas definições de robô, mas uma delas que se bem encaixa no contexto atual é um sistema autônomo que existe em meio físico, pode sentir o ambiente e agir sobre ele para atingir alguns objetivos (MATARIĆ, 2007).

Um tipo de robô mais próximo da nossa realidade é o robô de serviço. Robôs de serviço são robôs que realizam tarefas específicas voltadas para as necessidades do ser humano ou, ainda, robôs que não têm uma aplicação industrial (SINGH, 2012 apud NASCIMENTO, 2013, p. 43). Dentre os robôs de serviço, temos: robôs que realizam limpezas em ambientes domésticos, como o *Roomba*; robôs que auxiliam no ambiente de ensino; que auxiliam na exploração de ambientes hostis para seres humanos devido às dimensões, à presença de materiais tóxicos ou radioativos, ou ainda muito distantes, como o *Sojourner*, robô que explorou Marte; e robôs que realizam coletas, entregas ou buscas que procuram por minas em ambientes de guerra, entregam correspondência em escritório, remédios em hospitais ou de diversos materiais, como os coletores de latas e de bolas de tênis, entre outros.

O robô é um sistema complexo composto de diversas camadas tanto de *hardware* quanto de *software*. Basicamente, seu sistema possui esses elementos: os sensores, os atuadores e o sistema para processar os dados obtidos (FERREIRA, 2006). Estas camadas formam a arquitetura de controle do robô. O projeto desta arquitetura está intimamente ligado ao seu propósito (tarefa a ser realizada) e às características do ambiente desta tarefa. As arquiteturas de controle estão estabelecidas nas arquiteturas deliberativas, reativas e híbridas. A arquitetura híbrida une as qualidades das outras duas arquiteturas e separa suas funções em módulos de planejamento, sensoriamento e ação (MURPHY, 2000). A modularidade de uma arquitetura é também importante para torná-la escalável. Entende-se por modularidade a independência de funcionamento de um módulo para os outros e entende-se por escalabilidade a proporção de desempenho promovida por módulo de sistema.

Com o rápido desenvolvimento da tecnologia (e diminuição do custo), atualmente é possível utilizar diversos dispositivos como parte construtiva no robô, como os *smartphones* e uma grande variedade de microcontroladores. A tendência de usar dispositivos móveis é devida ao seu poder computacional, quantidade de sensores embarcados e facilidade na comunicação, levando em conta

seu custo. Com isto, é possível construir arquiteturas modulares poderosas e versáteis no controle de um robô móvel. Pode-se explorar o potencial dos microcontroladores nas tarefas de baixo nível, gerenciando os sensores e atuadores embarcados, e as atribuições da manutenção de sobrevivência. Como parte do sistema, um dispositivo móvel (*smartphone* ou *tablet*) é embarcado e pode usar de comunicação *Bluetooth* para se comunicar com o microcontrolador também embarcado. Além disso, usando a comunicação por rede sem fio, o dispositivo ganha acesso a bases computacionais mais poderosas (*desktops*) e à internet, podendo fornecer dados para que uma base possa processá-los e retornar informações relevantes para completar seu objetivo.

Arquiteturas construídas com estes recursos oferecem uma plataforma de desenvolvimento eficiente e suportam desenvolvimento numa ampla gama de aplicações, como o robô móvel *Roburguer* (Figura 1) desenvolvido pelo GISDI (Grupo de Integração de Sistemas e Dispositivos Inteligentes) da Universidade Estadual Paulista Júlio de Mesquita Filho no campus de Bauru, que pode – por exemplo – detectar faces e interagir verbalmente. Com foco em robôs que realizam exploração de ambientes com o intuito de procurar por objetos e coletá-los, o desenvolvimento deste trabalho será norteado por estes aspectos.

Figura 1 – *Roburguer* e seu dispositivo móvel.



Fonte: RANIERI, 2013, p. 22.

A plataforma se estabelece, por um lado, pela flexibilidade oferecida pela arquitetura ao desenvolvedor e, por outro, pela similaridade do tipo de serviço realizado pelo robô. Procurar por coisas (latas, bolas de tênis, minas terrestres), reconhecer uma área (exploração de planeta, áreas de catástrofe) ou de entrega (remédios, correspondência) entre outras situações similares, caracterizam exigências comuns e que podem ser cobertas por ambiente como este. Assim, um robô desenvolvido para uma determinada tarefa pode ser modificado, com menor esforço, para outras suportado por uma plataforma como a proposta.

1.1 OBJETIVO

O objetivo do projeto é implementar uma arquitetura de controle de *hardware* e de *software* para um robô de serviço que o torne mais eficiente e versátil para realização de tarefas com o uso de técnicas de visão computacional, exemplificado numa aplicação de coletar bolas de tênis em um ambiente.

1.2 PROPOSTA

Este é um trabalho de pesquisa e desenvolvimento de um robô móvel explorador de ambientes que, utilizando de técnicas de visão computacional, busca e coleta objetos. A arquitetura de *hardware* é baseada em microcontroladores para a camada inferior, responsável pela leitura dos sensores e o acionamento dos atuadores; por um sistema computacional, também embarcado, baseado em um dispositivo móvel; e por meio dos sistemas de comunicação disponíveis, enviar e receber informações de uma base computacional remota e mais poderosa. Certamente, a arquitetura de *software* segue o paradigma de três camadas, onde a camada mais baixa se preocupa com a navegação e a sobrevivência do robô no ambiente, a segunda, intermediária, permite tomadas de decisões locais e a camada superior é responsável pelo processamento intensivo e acesso a base de dados, permitindo a execução do plano da missão. Isto estabelece o ambiente de desenvolvimento e o trabalho de sua arquitetura de controle (vide capítulo 5).

1.3 MATERIAL E METODOLOGIA

1.3.1 Material

Neste trabalho foi desenvolvido o robô *Meg-alie*. Também foram necessários um *notebook Macbook Pro* com o ambiente *OpenCV* instalado e um *smartphone Samsung Galaxy Y* responsável por capturar imagens do ambiente, transmiti-las para o notebook e se comunicar com o robô e o computador.

Neste trabalho, o robô *Meg-alie* é a camada mais baixa da arquitetura de controle e cabe a ele se comunicar, via *bluetooth*, com o dispositivo móvel para receber as instruções que deve seguir, como rotacionar e seguir em frente, e enviar informações como a velocidade de rotação dos motores e os dados dos sensores. Detalhes sobre o robô podem ser lidos no capítulo 6.

O dispositivo móvel é a camada intermediária e é responsável por obter dados da câmera e

se comunicar tanto com a *Meg-alie* (via *bluetooth*) quanto com o computador (via rede). Além disso, caso não esteja conectado em uma rede, ainda assim é capaz de controlar o robô.

O computador externo é a camada mais alta da arquitetura e é responsável pelo processamento mais intenso dos dados obtidos pelo robô. Com ele é possível explorar recursos da computação gráfica que possam fornecer informações importantes sobre o ambiente de trabalho do robô.

1.3.2 Metodologia

- Investigação bibliográfica sobre técnicas de navegação de robôs móveis, de arquiteturas de controle e de técnicas de visão computacional que podem ser utilizadas neste projeto;
- Planejamento da arquitetura de *hardware* do robô
- Testes dos módulos isolados do robô;
- Montagem do robô;
- Testes dos módulos funcionando em conjunto;
- Implementação do sistema de comunicação *Bluetooth* entre o robô e o dispositivo móvel;
- Desenvolvimento da aplicação de captura de imagens do dispositivo móvel;
- Desenvolvimento da comunicação entre dispositivo móvel e computador externo;
- Tratamento da imagem transferida entre os dispositivos;
- Aplicação da visão computacional sobre a imagem;
- Planejamento de um algoritmo para a coleta de objetos pelo robô;
- Desenvolvimento de testes.

1.4 ORGANIZAÇÃO DA MONOGRAFIA

Esta monografia está dividida em oito capítulos e está ordenada da seguinte forma:

- O capítulo 2 trata sobre a eletrônica embarcada e equipamentos envolvidos necessários para o projeto do robô, como sensores e atuadores; microcontrolador embarcado e desenvolvimento de *software* embutido no dispositivo móvel (*smartphones*), para acesso aos

sensores (câmera) do sistema *Android*;

- O capítulo 3 dá uma breve introdução sobre os protocolos de rede de comunicação; visando a comunicação entre o dispositivo móvel e a eletrônica embarcada e, também, entre o dispositivo móvel e um computador remoto;
- O capítulo 4 descreve os conceitos e técnicas de visão computacional estudadas para realização do trabalho;
- O capítulo 5 trata da teoria sobre robótica necessária para que fosse alcançado o resultado atual do trabalho, expondo as principais arquiteturas de controle existentes; como representar um robô não-holonômico no ambiente; e as principais técnicas de navegação;
- O capítulo 6 explica sobre o sistema desenvolvido ao longo do projeto do ponto de vista dos três módulos propostos: o robô, o dispositivo móvel e o computador externo;
- O capítulo 7 descreve os testes e mostra os resultados obtidos ao longo do trabalho;
- O capítulo 8 apresenta a conclusão deste trabalho.

2. SISTEMAS EMBARCADOS

Neste capítulo serão abordados os conteúdos básicos para a compreensão do robô em sua camada de *hardware*.

2.1 ELETRÔNICA EMBARCADA

O robô é constituído de *hardware* e *software* e, de acordo com a definição dada no capítulo 1, ele tem a necessidade de sentir, raciocinar e agir sobre o sistema de acordo com seus objetivos. Dada a necessidade do uso de sensores, tanto para a navegação do robô pelo ambiente, para a verificação do movimento e do estado do robô, quanto para a identificação de objetos no ambiente, é necessário a determinação de quais e como estes sensores funcionam. Existem muitos sensores e para os mais variados propósitos e portanto, abordamos somente aqueles que são de interesse deste projeto. Por outro lado, o movimento do robô pelo ambiente e o seu propósito colaboram na escolha do sistema de tração necessário para a base móvel e assim, exigindo o conhecimento, funcionamento e o controle, dos tipos de motores disponíveis. Por outro lado, o movimento do robô pelo ambiente, caracterizando o sistema de tração da base móvel, exige o conhecimento, funcionamento e o controle, dos tipos de motores disponíveis. Por fim, o tratamento de como o fluxo de dados ocorrerá pelas camadas de *software*, tomadas de decisão e atuação sobre o ambiente e da missão (tarefa, missão e objetivo) determina a arquitetura de controle e o seu funcionamento.

2.1.1 Sensores

Os sensores são componentes eletrônicos que captam informações do ambiente e os representam por meio de sinal elétrico. Eles possuem muitas utilidades, entre elas a de identificar objetos e obstáculos no ambiente, além de medir a distância entre eles e o robô, o que é útil para que o robô se localize. Este tópico descreve sensores úteis para medir distância dos obstáculos e objetos, os quais ajudam no processo de coleta do objeto pelo robô.

O sensor infravermelho (Figura 2) é muito empregado em projetos para a detecção de objetos próximos e distantes do robô, mas, também, o uso de emissão e recepção de luz infravermelha pode ser empregada para outras aplicações, como a de determinação de velocidade de rodas em movimento ou, ainda, verificar se o robô está nos limites de um ambiente. O sensor precisa ser calibrado para cada ambiente de atuação, visto que a luminosidade pode gerar ruídos. Este sensor é constituído, basicamente, por um LED emissor e um receptor infravermelho. Este sensor não foi utilizado na montagem do robô, mas poderia substituir ou complementar o sensor ultrassônico embarcado no robô do sistema.

Figura 2 – Sensor de distância infravermelho



Fonte: Laboratório de Garagem.

Disponível em <<http://www.labdegaragem.org/loja/sensor-de-proximidade-infravermelho-sharp-gp2y0a21yk.html>>. Acesso em nov. 2015.

O sensor de distância ultrassônico (Figura 3) foi o sensor usado no robô *Meg-Alie*. Ele pode determinar com precisão a distância de objetos alinhados a ele e pode ser útil para saber o quanto o robô deve se locomover para se aproximar do objeto ou, ainda, saber se ele já está próximo o suficiente para ser empurrado. Um exemplo de sensor ultrassônico é o HC-SR04 e funciona da seguinte maneira: ele emite uma onda ultrassônica e, enquanto ela não é recebida, o sensor mantém a saída *ECHO* ativa e cabe ao controlador incrementar o tempo de espera pelo retorno da onda emitida enquanto ela não for recebida. Assim que ela voltar para o sensor, a saída *ECHO* é desativada e, assim, o controlador para de incrementar o tempo de espera. Tendo o tempo de espera e conhecendo a velocidade da onda ultrassônica em seu ambiente de atuação, é possível obter a distância percorrida através da fórmula:

$$S = \frac{V \cdot t}{2} ,$$

em que *S* é a distância entre o sensor e o objeto, *V* é a velocidade do som e *t* é o tempo de espera.

Figura 3 – Sensor de distância ultrassônico HC-SR04



Fonte: elaborado pelo autor.

2.1.2 Atuadores

Atuadores são responsáveis pela forma como o robô pode agir sobre o ambiente.

Normalmente são associados à locomoção. Nesta seção são descritos dois atuadores: o motor de corrente contínua e o servo-motor.

O motor de corrente contínua, dado uma tensão e uma corrente, será ativado com determinada velocidade de rotação e torque. Normalmente esses motores são acoplados às caixas de redução para que haja um aumento no torque e menor velocidade de giro, possibilitando a movimentação do robô pelo ambiente. Entretanto, seu uso normalmente não é tão simples, visto que é comum que a corrente de saída de um microcontrolador seja insuficiente para servir de alimentação para o motor, necessitando amplificação da corrente, isto é, de uma interface de potência. Quando a corrente do microcontrolador é ampliada somente por meio de transistores e eles estão num arranjo semelhante a uma ponte, permite também a inversão do sentido de rotação dos motores. Assim, a solução é o uso de uma ponte-H – circuito mais complexo que, como dito, utiliza de um arranjo de transistores que permite a amplificação, a inversão do sentido e alterações na velocidade de giro.

O servo-motor tem sua rotação controlada por pulsos de sinal. A largura do pulso (em unidade de tempo) define o ângulo final. Entretanto, é necessário que o pulso seja repetido em uma frequência de 20ms por um número determinado de vezes para que dê tempo do eixo alcançar o ângulo de rotação desejado. Basicamente, a largura do pulso para manter o eixo em 90° é de 1,5ms. Para que o eixo se movimente para 0°, os pulsos precisam ter o comprimento de 1ms. Por fim, para que o eixo fique a 180°, os pulsos precisam ter o comprimento de 2ms. A Figura 4 mostra a relação entre a largura do pulso e o deslocamento angular do eixo do motor.

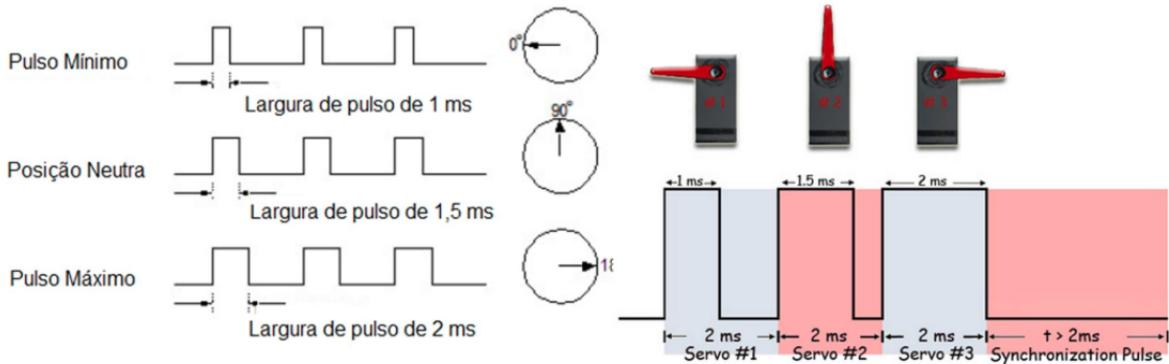
Com isso, é possível definir uma equação básica do comprimento do pulso em função do ângulo desejado:

$$t = \frac{\theta}{180} + 1 ,$$

em que t é o tempo em milissegundos e θ é o ângulo desejado em graus.

Inicialmente, o servo-motor seria utilizado para rotacionar o sensor de distância ultrassônico do robô numa varredura de 0 a 180°, para ampliar a área de busca e obter a distância de objetos no ambiente. Entretanto, ele acabou não sendo utilizado no robô pois aumentaria ainda mais seu custo e complexidade. Seu uso foi substituído por simples rotações da base do robô pelo seu eixo no ângulo desejado.

Figura 4 – Funcionamento do servo-motor.



Fonte: Aula de engenharia elétrica da UNESP. Disponível em:

<<http://www.feis.unesp.br/Home/departamentos/engenhariaeletrica/aula-4---servo-motor-13-03-2013-final.pdf>>. Acesso em ago. 2015.

2.1.3 O microcontrolador embarcado - *Arduino*

Um microcontrolador é o “cérebro” do módulo de controle do robô. Ele é um circuito responsável por processar e tratar os dados obtidos pelos sensores, bem como controlar os atuadores de acordo com seus planos. Atualmente, os microcontroladores PIC, da *Microchip*, e os da *ATMEL* são muito populares para sistemas embarcados. O PIC ainda depende de equipamento de alto custo para ser programado e, como ele precisa ser soldado à uma placa para ser usado, a manutenção dos programas e correção do código se torna muito mais complexa, visto que é preciso dessoldar o microcontrolador da placa sempre que precisar reprogramá-lo. Já os da *ATMEL* estão cada vez mais populares por terem sido integradas nas placas *Arduino*, que são placas que possuem todos os componentes mínimos para o funcionamento do *ATMEL* soldados. Para carregar o programa para o microcontrolador basta conectar o cabo da placa para o computador e executar o processo pela IDE (do inglês, Ambiente de Desenvolvimento Integrado). Por ser muito simples de programar e por ter todos os componentes mínimos acoplados, o *Arduino* pode ser inserido em projetos por pessoas que não têm um conhecimento aprofundado em programação ou eletrônica.

Existem diversos modelos de *Arduino* no mercado, desde modelos pequenos, do tamanho de uma moeda, até modelos completos, com diversos sensores como giroscópio e acelerômetro embutidos. Alguns destes modelos estão representados na Figura 5.

Outra vantagem no *Arduino* está na modularidade dos projetos: normalmente existem sensores adaptados para o *Arduino* (também chamado de *shields*) que possuem uma biblioteca própria e otimizada para trabalhar com eles. Alguns *shields* já são adaptados de tal forma que basta encaixá-los no *Arduino* para estarem funcionando.

Para programar um *Arduino*, é preciso obter a IDE no site da fabricante www.arduino.cc. No

caso do sistema operacional *Windows* é só executar o instalador ou extrair o arquivo compactado. Nos dois casos a instalação é intuitiva. No *Linux* é possível instalar pelo *apt-get* (ou comando equivalente) e, no *Mac OS X*, basta extrair o *dmg* na pasta “Aplicações”.

Figura 5 – Diferentes modelos de Arduino



Fonte: ArdUFC. Disponível em: <<http://ardufc.blogspot.com.br/2012/10/modelos-do-arduino.html>>.

Acesso em nov. 2015.

A interface da IDE, apresentada na Figura 6, é minimalista e a linguagem de programação é uma adaptação da linguagem *C++*. Entretanto, ao invés do método **main** conhecido na linguagem *C* e *C++*, o *Arduino* implementa dois métodos: o **setup** e o **loop**, ambos do tipo *void*.

O método **setup** é o primeiro a ser executado e ele é responsável por realizar as atribuições iniciais e definir as portas de entrada e saída. Quando o método **setup** termina, é iniciado o método **loop**, que é um laço onde o *Arduino* pode obter dados dos sensores, processar informações, se comunicar com outros dispositivos e controlar os atuadores.

Figura 6 – IDE Arduino



Fonte: elaborado pelo autor.

2.2 O SMARTPHONE

Smartphones estão cada vez mais presente no dia-a-dia. Estes dispositivos possuem grande poder de processamento e diversos tipos de sensores, além de câmera. Com a redução do custo e com a facilidade de implementação destes dispositivos, tem se tornado interessante a utilização destes em projetos de robótica.

De acordo com a *International Data Corporation* (2015), os 3 sistemas operacionais para *smartphones* mais populares são, respectivamente, o *Android*, o *iOS* e o *Windows Phone*. O *Android* é um sistema de baixo-custo de aquisição, possui grande quantidade de API (do inglês, Interface de Programação de Aplicativos) e serviços disponíveis, sem contar que não precisa pagar por uma licença de desenvolvedor. Para desenvolver para o sistema *iOS* só é possível através de computadores da Apple e ainda é preciso pagar por uma conta de desenvolvedor caso seja desejado publicar o aplicativo; e para desenvolver para *Windows Phone* é preciso pagar pela conta de desenvolvedor para registrar e usar o dispositivo móvel para os testes. Por fim, como o sistema *Android* é mais acessível e menos burocrático, ele foi escolhido para este projeto.

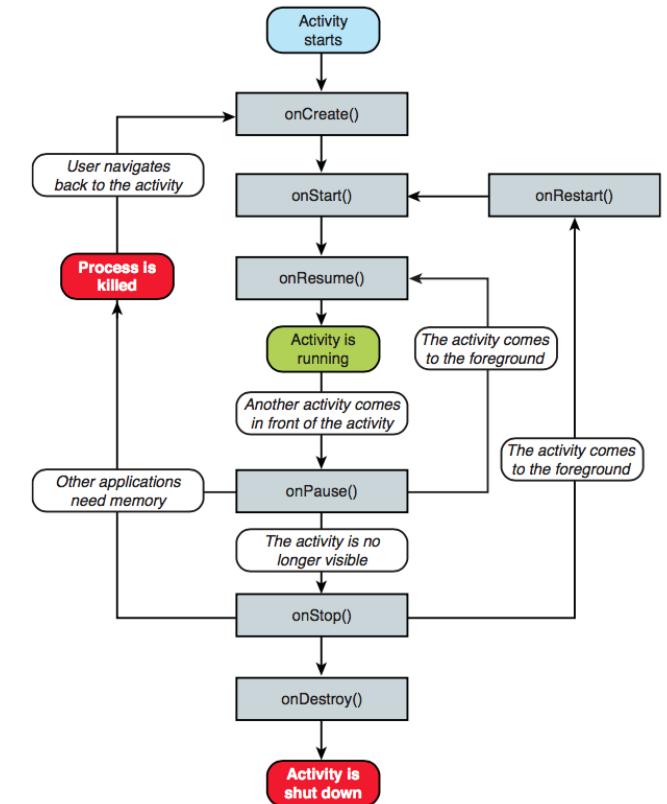
As aplicações *Android* possuem 4 componentes básicos essenciais (SHWARZ et al., 2013): *Activity*, *Service*, *BroadcastReceiver* e *ContentReader*. Cada um deles está descrito na Tabela 1.

Esses componentes (com exceção do *ContentReader*) são ativados por uma mensagem assíncrona chamada *Intent*. Esta possui um *Bundle* (pacote) com informações que descrevem o componente. Isto provê método de transferência de informação entre os componentes. O ciclo de vida de uma *Activity* é representada pela Figura 7. Note que nem todos os métodos precisam ser sobreescritos ao se desenvolver uma aplicação *Android*.

Tabela 1 – Componentes básicos para programação *Android*.

Componente	Descrição
<i>Activity</i>	Toda aplicação possui pelo menos uma <i>Activity</i> . É o processo principal da aplicação.
<i>Service</i>	É um serviço que fica em segundo plano. Pode ser um serviço que atualiza o e-mail ou que reproduz música enquanto a tela do dispositivo está desligada.
<i>BroadcastReceiver</i>	Responsável por receber eventos; gatilho.
<i>ContentReader</i>	Acessa os recursos do telefone, tais como a lista de contatos.

Fonte: elaborado pelo autor.

Figura 7 – Ciclo de vida de uma aplicação *Android*.

Fonte: SHWARZ et al., 2013, p. 31.

Como já foi dito, uma das grandes vantagens dos *smartphones* é o fácil acesso aos componentes de *hardware* e sua fácil implementação. Dentre esses componentes estão a câmera e os sensores disponíveis, ambos serão vistos nos dois próximos tópicos.

2.2.1 Acesso à câmera do smartphone

A câmera é um dos acessórios de um dispositivo móvel mais atraente para os seus usuários. Com a evolução dos *smartphones*, as câmeras estão cada vez mais poderosas e acessíveis. Elas podem ser usadas para armazenar imagens do cotidiano ou filmar um ambiente. É possível, também, usá-las para obter dados por meio de visão computacional aplicada às imagens obtidas, o que é de grande utilidade para o projeto em questão.

Nos dispositivos *Android*, a câmera possui uma interface muito simples e pode ser trabalhada com duas bibliotecas: uma para dispositivos mais antigos (mas que funciona nos mais novos), chamada **Camera**, e a outra – para dispositivos mais atuais – chamada **Camera2**. Como o dispositivo usado para este trabalho possui a API 10 (obsoleta) do *Android*, **Camera** será abordada.

Para usar recursos do sistema *Android*, é preciso alterar um arquivo de permissões no projeto chamado “*AndroidManifest.xml*”. A câmera é um recurso que exige ser declarado neste arquivo, e para fazer isto basta adicionar o trecho de código:

```
<uses-permission android:name="android.permission.CAMERA" />
```

A obtenção da imagem em tempo real da câmera utiliza de três classes: **Camera**, que tem acesso ao *hardware* da câmera; **Camera.Parameters**, que possui todas as configurações da câmera; e **SurfaceView**, que provê de uma superfície que projeta a imagem da câmera em tempo real. Além disso, é preciso implementar uma classe **SurfaceHolder.Callback**, que gera ações para mudanças no **SurfaceView**.

O **SurfaceView** implementa três métodos:

- *surfaceCreated()*, que executa quando a superfície é criada pela primeira vez;
- *surfaceChanged()*, chamada após a superfície ser criada ou quando alguma propriedade é alterada;
- *surfaceDestroyed()*, que é chamada entre a remoção da superfície da **View** e a destruição da superfície.

A classe **Camera** tem como método de tratamento para cada frame do *Preview* (imagem do frame atual da câmera) o *setPreviewCallback()*, este tem como parâmetro uma classe **Camera.PreviewCallback** que implementa o método *onPreviewFrame*. Neste método é possível acessar o vetor de bytes da imagem (que está no formato YUV¹), que pode ser convertido para qualquer outro formato em outro momento.

2.2.2 Acesso aos sensores do smartphone

Os dispositivos móveis com *Android* possuem diversos tipos de sensores. De acordo com

¹ Modelo de representação da cor dedicado ao vídeo analógico.

SHWARZ et al. (2013), os dois sensores padrão dos *smartphones Android* são o acelerômetro de três eixos e o magnetômetro de três eixos. Na Tabela 2 estão descritos alguns dos sensores disponíveis para os dispositivos *Android*.

Tabela 2: Alguns sensores disponíveis para *Android*.

Tipo	Descrição
TYPE_ACCELEROMETER	Mensura a aceleração em metros/segundo
TYPE_ALL	É uma constante que descreve os tipos de sensores
TYPE_GYROSCOPE	Mede a orientação baseado no momento angular
TYPE_LIGHT	Mensura a luz ambiente em lux
TYPE_MAGNETIC_FIELD	Mensura o campo magnético em micro-Tesla
TYPE_PRESSURE	Mensura a pressão do ar
TYPE_PROXIMITY	Mede a proximidade de um objeto em centímetros
TYPE_TEMPERATURE	Mede a temperatura em graus Celsius

Fonte: SHWARZ et al., 2013, p. 176.

O acesso aos sensores se restringe, basicamente, às classes **Sensor**, **SensorManager** e **SensorEventListener**. Com a classe **SensorManager** é possível obter instâncias dos sensores (classe **Sensor**) pelo método *getDefaultsensor*, que recebe como parâmetro a constante do sensor desejado. Assim que instanciado, é preciso atribuir ao sensor um **SensorEventListener**, que é a classe que implementa os métodos que ocorrem quando há qualquer mudança no sensor ou em suas configurações. Dentre esses métodos tem-se o *onSensorChanged*, que tem como parâmetro um objeto de **SensorEvent**. Este fornece o tipo de sensor que sofreu a mudança e os valores do sensor no vetor *values*.

Com estes sensores é muito mais fácil de fazer com que o robô possa se localizar pelo ambiente, podendo – assim – se utilizar de técnicas de navegação descritas no capítulo 5.

3. COMUNICAÇÃO ENTRE OS SISTEMAS

Dada a arquitetura de *software* do sistema robótico, é interessante que o dispositivo móvel presente na camada intermediária se comunique com o computador da camada superior ou com o robô da camada inferior por meio da rede. Por se tratar de um esquema de comunicação entre dois pontos (P2P), foram estudados, para a camada de aplicação, os protocolos de comunicação UDP (do inglês, Protocolo de Datagrama do Usuário) e TCP (do inglês, Protocolo de Controle de Transporte).

No processo de transmissão de dados, os dados são divididos em unidades menores de tamanho fixo, chamados datagramas. Estas unidades possuem duas áreas: a informação propriamente dita e um cabeçalho, que contém informações relevantes para a transmissão.

O UDP é um protocolo que envia datagramas IP sem que seja necessário estabelecer uma conexão. Ele não realiza controle de erros ou retransmissão de dados ao receber um segmento incorreto, cabendo à aplicação de usuário tratar a informação recebida (TANENBAUM, 2002). Por se tratar de um protocolo simples e que não é orientado à conexões, é considerado um protocolo rápido, por isso é muito utilizado para streaming de vídeos, por exemplo.

O TCP foi projetado para oferecer um fluxo de bytes fim-a-fim confiável (TANENBAUM, 2002). Como a camada IP não garante que os datagramas serão entregues na ordem correta, por exemplo, cabe ao TCP ordená-los e retransmiti-los sempre que necessário.

Tendo estas informações em vista, o protocolo utilizado para a transmissão de imagens entre a camada intermediária e a camada superior foi o TCP, visto que a imagem não pode sofrer qualquer alteração no processo de transmissão.

Entretanto, para a transmissão de dados da camada inferior para a camada intermediária foi utilizada a tecnologia *bluetooth* e o protocolo utilizado foi outro desenvolvido pelo autor e consiste, basicamente, de troca de dados de 1 byte. Esse protocolo é explicado com detalhes no capítulo 6 quando será apresentada a camada inferior da arquitetura.

4. VISÃO COMPUTACIONAL

A visão humana é um sentido muito útil. Com ela, podemos estimar distância entre os objetos, tamanho, luminosidade em um ambiente e saber se algo está se movendo, por exemplo. A visão nos permite obter informações do ambiente no qual estamos inseridos. Em termos computacionais, a visão também é responsável por captar informações em uma imagem. A definição de Szeliski (2010) diz que a visão computacional tenta descrever o mundo que vemos nas imagens e suas propriedades.

Neste capítulo serão abordados alguns conceitos e técnicas fundamentais de visão computacional usados para a elaboração deste projeto.

4.1 REPRESENTAÇÃO DE CORES

A imagem é constituída de diversos pixels das mais variadas cores. Por meio da visão, sabemos definir se um pixel é azul, roxo ou laranja, por exemplo. Assim, é importante entender a análise e a representação digital de uma cor.

Existem diversas formas de realizar essa representação, dentre elas tem-se a RGB (*Red, Green, Blue*), e HSV (*hue, saturation, value*).

O RGB é uma estrutura que possui a quantidade de vermelho, verde e azul em um pixel. Essas cores são cores primárias aditivas, ou seja, a combinação entre elas gera cores tais como na Figura 8.

Figura 8 – Cores aditivas



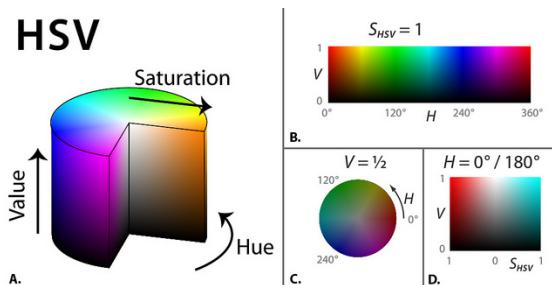
Fonte: SZELISKI, 2010, p. 81.

O cilindro HSV, ilustrado na Figura 9(a), basicamente, representa uma cor através do *hue* (ou matiz), que é definida como o ângulo de direção do corte horizontal do cilindro de cores com relação ao vermelho puro, conforme a Figura 9(c); da *saturation* (do inglês, saturação), que é calculada com relação ao vetor de luminosidade² e a distância do centro da roda de cores, que é

² É um outro elemento usado na representação de cores e envolve uma série de transformações.

representada na Figura 9(d); e do *value* (do inglês, valor), ilustrado na Figura 9(b), que determina a intensidade mínima ou máxima da cor (HANBURY, 2002; SZELISKI, 2010).

Figura 9 – HSV cilíndrico



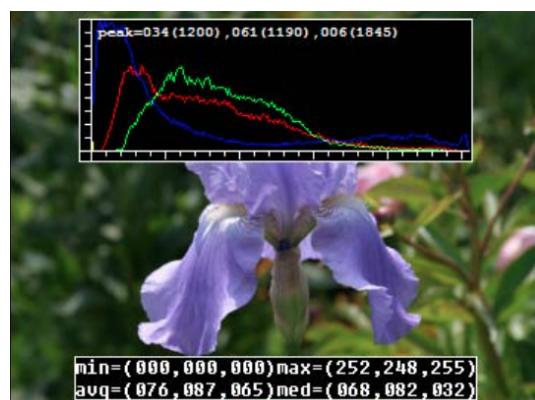
Fonte: HANZRATECH. Disponível em: <<http://hanzratech.in/2015/02/07/caveat-thresholding-hue-component.html>>. Acesso em 20 jan. 2016.

4.2 HISTOGRAMAS

Um histograma é uma representação de distribuição de dados. Nesta distribuição, os dados são agrupados em intervalos de valores. Assim, o histograma indica a frequência de que um determinado intervalo de valor ocorre.

No contexto da computação gráfica, os histogramas podem ser usados tanto para a análise quanto para a equalização da imagem. Um caso de uso prático, por exemplo, é detecção de objetos por cor. A Figura 10 representa uma imagem com seu histograma e alguns dados como o valor mínimo, máximo, médio e qual é ponto de pico de cada curva. É possível observar que a curva do histograma das três cores se concentra nos valores de menor intensidade. Isto acontece porque a imagem possui tons escuros, o que implica em valores menores de vermelho, verde e azul.

Figura 10 – Exemplo de distribuição de cores de uma imagem.



Fonte: SZELISKI, 2010, p. 102.

4.3 DETECÇÃO DE BORDAS

Quando enxergamos um objeto sabemos onde ele começa e onde ele termina. Sabemos disto pois, ao olharmos para um ambiente, naturalmente detectamos as bordas dos objetos que pertencem a ele.

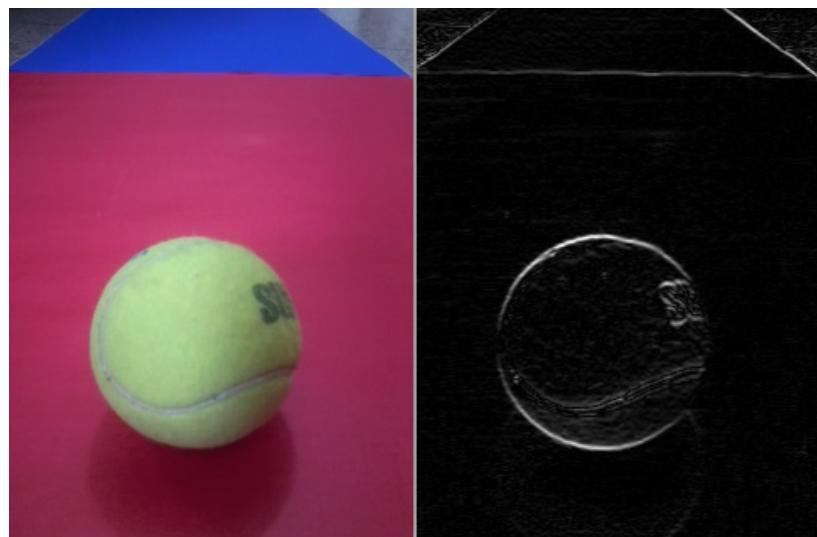
De acordo com Szeliski (2010), uma borda em uma imagem é uma rápida variação de intensidade. Colocando em termos matemáticos, de forma geral, a inclinação e a direção de intensidade de uma superfície é definida por

$$\mathbf{J}(\mathbf{x}) = \left(\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y} \right)(\mathbf{x}) ,$$

em que \mathbf{J} é o vetor de gradiente que aponta para a direção de subida mais acentuada na função de intensidade de uma imagem \mathbf{I} .

O operador de **Sobel** é um dos operadores usados para calcular uma aproximação das derivadas parciais de uma imagem. O cálculo é realizado através de uma matriz de convolução³ e aproxima a derivada para uma função polinomial de segundo grau. (BRADISKI; KHAELER, 2008). A Figura 11 mostra o efeito do operador de Sobel aplicado na direção do eixo y (vertical).

Figura 11 – Imagem original (esquerda) e efeito do operador de Sobel aplicado na direção y (direita).



Fonte: elaborado pelo autor.

Além do Sobel, outro método usado para encontrar as bordas de uma imagem é o de **Canny**. Este método considera que, dada uma imagem cuja convolução Gaussiana já tenha sido aplicada para redução de ruídos, a borda é o pixel cuja derivada segunda é zero. A equação que representa

³ Convolução é uma operação que é realizada com cada parte da imagem. Esta operação utiliza de um vetor de coeficientes chamada núcleo de convolução (BRADISKI; KHAELER, 2008).

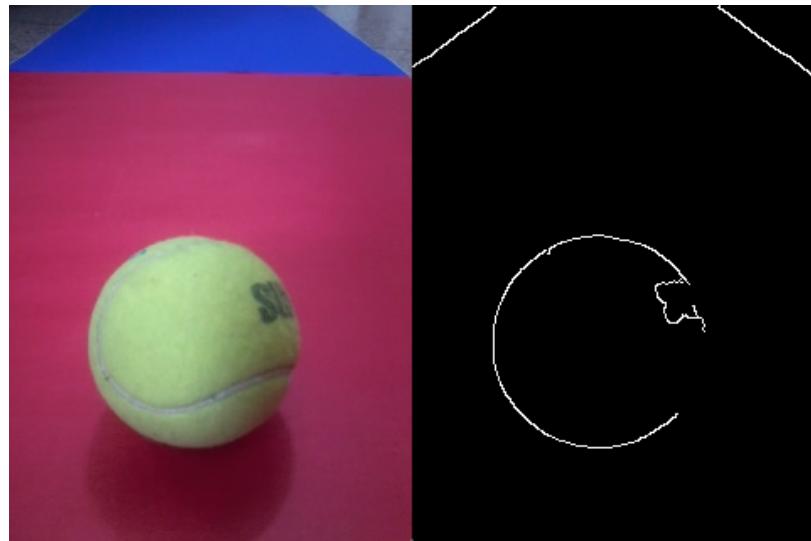
este conceito é:

$$\frac{\partial^2}{\partial n^2} G * I = 0 \quad ,$$

em que n é a direção desejada, G é a matriz Gaussiana, o operador $*$ denota convolução e I é a imagem.

Além disso, este método realiza um limiar de histerese, ou seja, são estipulados um limite inferior e um limite superior. Para cada pixel, se a intensidade do gradiente é menor que o limite inferior, este pixel é rejeitado como borda. Se a intensidade do gradiente é maior que o limite superior, o pixel é aceito como borda. Por fim, se o pixel se encontra entre os limites, este só irá ser considerado como borda se está conectado com outra borda (BRADISKI; KHAELER, 2008; CANNY, 1986). A Figura 12 é um exemplo implementado em C++ usando o método *Canny* da biblioteca *OpenCV* com limite inferior igual à 40 e limite superior igual à 120.

Figura 12 – Imagem original (esquerda) e detecção de bordas de Canny (direita).



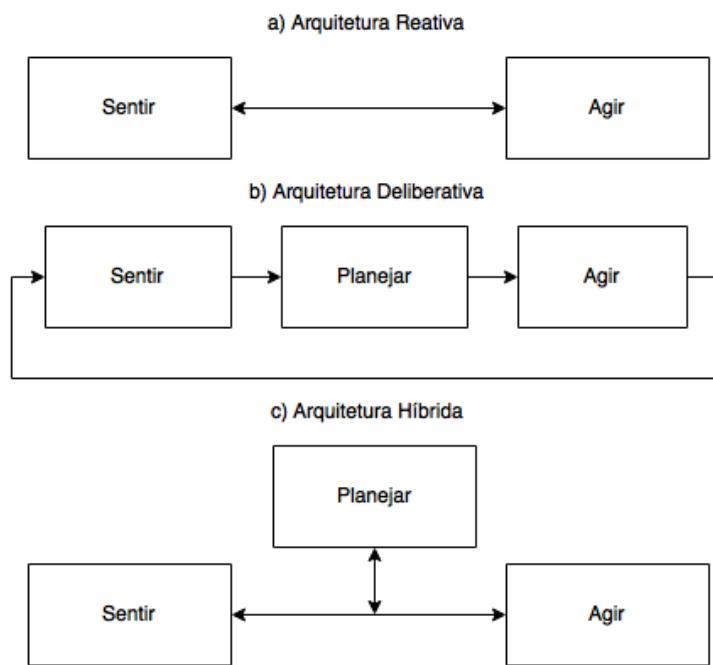
Fonte: elaborado pelo autor.

5. ROBÓTICA MÓVEL AUTÔNOMA

5.1 ARQUITETURA DE CONTROLE

A arquitetura de controle é responsável pela forma como o robô irá agir ou pensar. Existem três tipos de arquitetura: a reativa, a deliberativa e a híbrida (MURPHY, 2000). A arquitetura reativa é a que para cada estímulo ou conjunto de estímulos exista uma reação. A arquitetura deliberativa consiste em sentir o ambiente, planejar uma ação a partir disto e executar uma ação de acordo com o planejamento. A arquitetura híbrida une a ideia das outras duas: basicamente, o agente híbrido primeiro planeja como decompor sua tarefa principal em tarefas menores. A partir daí o agente irá se comportar como um agente reativo, ou seja, vai possuir um conjunto de ações para cada estímulo dado pelos sensores de acordo com o que foi previamente planejado. A Figura 13 apresenta, em blocos, os esquemas das arquiteturas descritas.

Figura 13 – Arquiteturas de Controle



Fonte: elaborado pelo autor.

Entretanto, segundo Mataric (2007), é preciso considerar uma série de fatores antes de decidir qual arquitetura de controle irá ser adotada, dentre eles:

- Quantidade de ruído do sensor no ambiente de trabalho;
- Se o ambiente é dinâmico ou estático;
- Se o robô é capaz de conhecer todas as informações do ambiente ou, senão, o quanto

ele é capaz de sentir;

- A rapidez com que o robô consegue sentir o ambiente;
- A rapidez com que o robô consegue agir;
- Quantidade de ruído nos atuadores;
- Se é preciso que o robô tenha consciência de eventos passados para tomar decisões;
- Se o robô precisa realizar previsões.

Ainda de acordo com Mataric (2007), as arquiteturas de controle podem ser observadas com foco em três atributos: tempo (ou seja, se as decisões visam ações de curto ou longo prazo); modularidade – o quanto os módulos da arquitetura de *software* do robô interagem entre si e o quanto independentes eles são; e representação, que se refere à forma que o robô armazena informação em sua memória.

Para o projeto, a arquitetura híbrida acabou sendo empregada, visto que o sistema realiza duas funções: buscar o objeto e levá-lo até um local final. Durante o período de busca, o plano é explorar o ambiente. Com isso, são definidas ações para os dados sentidos durante exploração (sistema reativo). Quando o robô encontrar o objeto, é planejado um trajeto que será realizado para chegar até ele e, depois disso, um trajeto que o levará até o local final (sistema deliberativo). Essa cooperação de sistema reativo e deliberativo caracterizam um sistema híbrido.

5.2 TRAÇÃO DIFERENCIAL

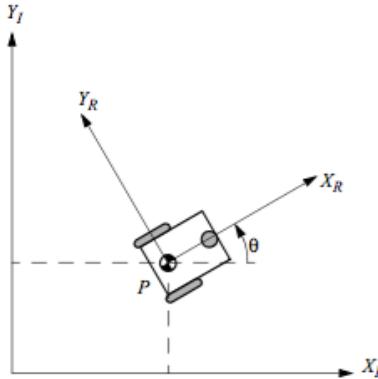
Um robô móvel precisa se locomover por um ambiente para poder explorá-lo e, assim, realizar suas tarefas para alcançar seus objetivos. Os robôs móveis podem rastejar, tracionar rodas, navegar ou voar, por exemplo, mas sua forma de locomoção está relacionada, principalmente, com o ambiente do seu objetivo.

Dentre os modelos de robôs de tração existentes temos os robôs holonômicos e robôs não-holonômicos. Os não-holonômicos são robôs que possuem alguma restrição de movimento quanto aos graus de liberdade, fazendo com que seja necessário a realização de manobras para se mover para um determinado ponto (SIEGWART; NOURBAKHS; SCARAMUZZA, 2004).

Dado o plano cartesiano (X_1, Y_1) com origem em O, dado um ponto qualquer local do robô P(X_R, Y_R), global Q(X, Y) e da sua orientação θ , com relação ao eixo X, podemos representar um quadro global por ξ_L , a matriz de rotação ortogonal por R_θ e o quadro local por ξ_R , em que:

$$\xi_I = \begin{bmatrix} X \\ Y \\ \theta \end{bmatrix}, \quad R_\theta = \begin{bmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \text{e} \quad \xi_R = \begin{bmatrix} X_R \\ Y_R \\ \theta_R \end{bmatrix} = \xi_I \cdot R_\theta .$$

Figura 14 – Representação do robô no plano cartesiano.



Fonte: SIEGWART; NOURBAKHSH; SCARAMUZZA, 2004, p. 49.

Entretanto, como este modelo não abrange situações mais complexas, temos que dado o diâmetro R das rodas de tração; o ponto P (X_R , Y_R) centralizado entre as duas rodas – cada qual com a distância L de P ; as velocidades de rotação das rodas ϕ_1 e ϕ_2 ; um modelo que faria a previsão da velocidade geral do robô seria:

$$\xi_I = R_\theta^{-1} \cdot \begin{bmatrix} X_R \\ 0 \\ \omega \end{bmatrix} \quad \text{e} \quad \omega = \frac{R \cdot \phi}{2 \cdot L} ,$$

em que ω é a soma da velocidade angular das duas rodas; ϕ_1 e ϕ_2 são informações fornecidas pelo fabricante do motor. Assim, ξ_I irá indicar a direção para onde o robô irá se mover.

5.3 TÉCNICAS DE NAVEGAÇÃO

A navegação está comprometida com o deslocamento seguro de um robô entre um ponto de partida e o ponto de chegada. Para este deslocamento é necessário informações sobre o ambiente e da localização do robô nele. Este problema foi resumido pelas questões: “Onde estou?”, “Para onde vou?” e “Como chego lá?” (LEONARD; DURRANT-WHYTE, 1991). Portanto, o posicionamento do robô no ambiente e a sua navegação para o seu ponto destino podem ser realizados por técnicas relacionadas a posicionamentos absoluto ou relativo. As técnicas de navegação estão baseadas nos sensores utilizados.

O posicionamento relativo está baseado no uso de odometria ou de navegação inercial. A odometria se apropria de medidores de rotação das rodas e o inercial de sensores como giroscópio e

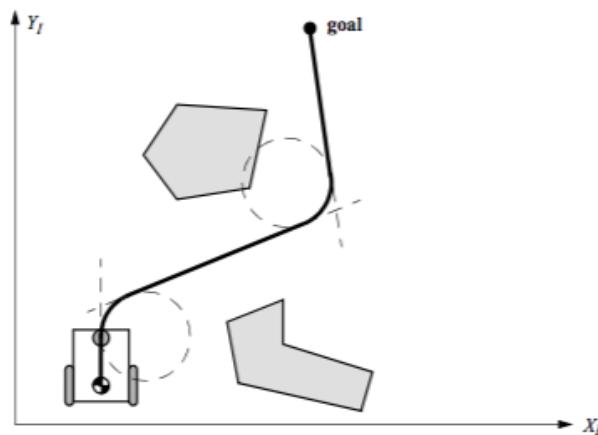
acelerômetros. Ambos se referenciam ao ponto de partida e erros acumulados podem afastá-los do objetivo durante a navegação.

Por outro lado, o posicionamento absoluto se apropria de informações de *Beacons* ativos (do inglês, faróis), como sinais de rádio ou luz transmitidos de locais de posicionamento conhecido, de *Landmarks* (do inglês, pontos de referência) naturais ou artificiais, como marcas colocadas no ambiente (por exemplo: setas e placas) ou de características do próprio ambiente (portas e prédios), respectivamente. Por fim, é a navegação baseada em mapas e no encontro de informações locais e globais.

Entre o ponto de partida e o destino pode ter obstáculos fixos ou móveis. Portanto, um plano de navegação deve permitir um replanejamento sempre que um imprevisto ocorrer. Por exemplo, a navegação baseada em mapas apresenta somente os caminhos ou obstáculos fixos porém, durante a navegação ele pode ter sido alterado e rotas alternativas devem ser planejadas e neste caso, somente com as informações locais.

Como exemplo, é a técnica chamada de *open loop*, também conhecida como *Trajectory-Following* (SIEGWARD et al., 2004). Ela consiste em dividir a trajetória em segmentos de movimento com formas claramente definidas como linhas retas e arcos de circunferência. O problema desta técnica é que não é fácil computar uma trajetória viável se todas as limitações e restrições de velocidade e aceleração forem levadas em conta. Além disso, o robô não corrige ou adapta sua trajetória se mudanças ocorrerem no ambiente (ambiente dinâmicos). Somado a tudo isso, como as trajetórias não são suaves, há descontinuidade na aceleração do robô.

Figura 15 – Exemplo de *Open Loop*.

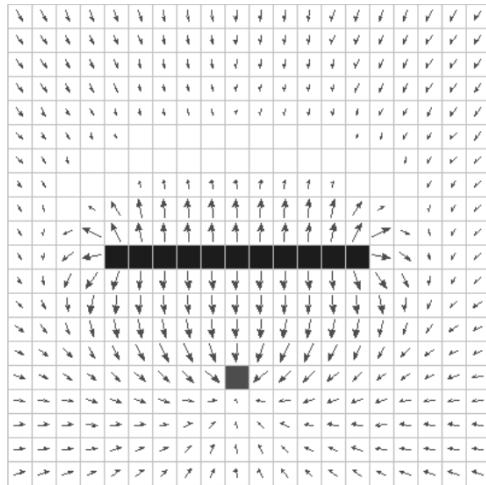


Fonte: SIEGWART; NOURBAKHSH; SCARAMUZZA, 2004, p. 82.

Outra técnica conhecida é a de **campos potenciais**. Segundo Silva (2011), esta consiste em

considerar obstáculos como forças repulsivas e o objetivo como força atrativa (tais como ocorre nos campos elétricos entre cargas positivas e negativas). É uma técnica que exige baixo custo computacional, mas só é possível de ser implementada em ambientes nos quais a posição do robô, dos obstáculos e do objetivo são conhecidas. A figura 16 exemplifica um campo potencial.

Figura 16 – Campos Potenciais.



Fonte: SILVA, 2011, p. 18.

O problema dessas duas técnicas é que o robô precisa ter visão de todo o ambiente ou um mapa para poder executá-las. Entretanto, a navegação também pode ser realizada de forma reativa usando, para isto, sensores, como a câmera. Neste caso, para cada imagem capturada há uma exploração da mesma na busca de indicadores que permitem a navegação. Certamente, misturas de técnicas são, normalmente, utilizadas e bem vindas.

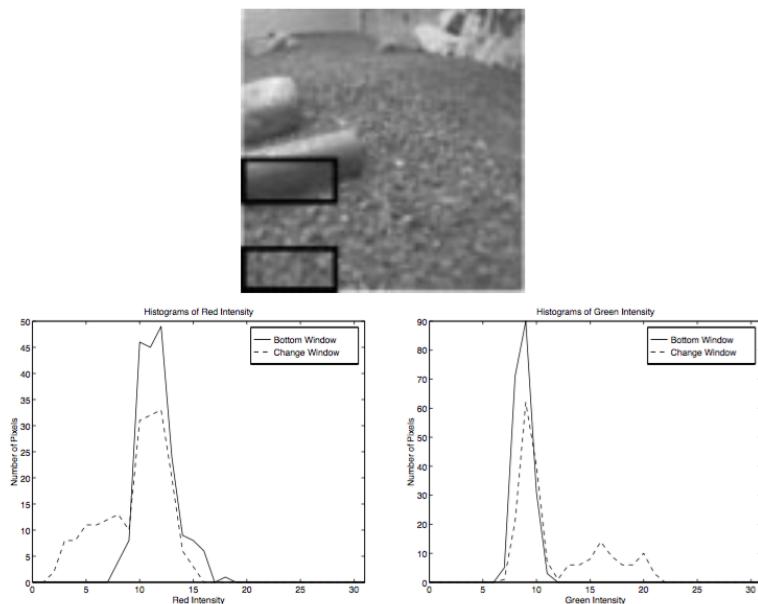
A técnica de navegação baseada em visão computacional proposta por Brooks, Grimson e Lorigo (1997) independe do conhecimento completo do ambiente, bastando somente da imagem obtida através de uma câmera acoplada ao robô. A técnica combina de três módulos de visão diferentes: o HSV, o RGB e o de detecção de bordas, sendo que cada um dos conceitos utilizados foi explicado no capítulo 4. Este método leva em consideração que os obstáculos do ambiente estão longe o bastante do robô para serem capturados na imagem e que o terreno próximo ao robô está livre de obstáculos. A idéia é basicamente comparar partes do terreno próximo ao robô com partes do terreno que tem possibilidade de ter obstáculos, sendo que as duas partes estão na mesma imagem. Caso haja alguma diferença grande entre os terrenos, ali existe um obstáculo.

Para comparar os dois terrenos, a imagem é separada em quadros de tamanho fixo menores que ela. Um dos quadros fica fixo na parte de baixo da imagem, que é do terreno próximo do robô e o outro quadro inicia acima do quadro fixo e se desloca na direção vertical a cada iteração.

Quando o quadro dinâmico alcança o topo da imagem, o quadro fixo se desloca na horizontal e o processo e o quadro dinâmico volta a se deslocar na vertical. Assim vai até que toda a imagem seja analisada.

A Figura 17 mostra os quadros de comparação, onde o de baixo é o quadro fixo e o superior é o quadro dinâmico. O histograma da esquerda representa a distribuição das intensidades de vermelho na imagem e o histograma da direita a das intensidades de verde na imagem.

Figura 17 – Comparação entre os quadros da imagem.



Fonte: BROOKS, GRIMSON e LORIGO, 1997, p. 375.

O processo de comparação entre os quadros consiste gerar o histograma de cada módulo em duas dimensões e encontrar a área entre os histogramas dos dois quadros. As distribuições usadas no módulo HSV são a intensidade da *hue* e a da saturação; no módulo RGB são a intensidade de vermelho e verde; e no módulo de detecção de bordas são a intensidade do gradiente na direção x e na direção y.

A área calculada entre os histogramas é dada pela seguinte fórmula:

$$\text{area} = \sum_v |h(v) - s(v)| ,$$

em que **h** é o histograma do quadro dinâmico, **s** é o histograma do quadro fixo e **v** é o valor da intensidade da distribuição escolhido.

6. DESENVOLVIMENTO DAS CAMADAS DO ROBÔ PROPOSTO

O desenvolvimento do sistema do robô foi dividido em três partes: desenvolvimento do robô (camada inferior), implementação de *software* do dispositivo móvel (camada intermediária) e o do computador externo (camada superior).

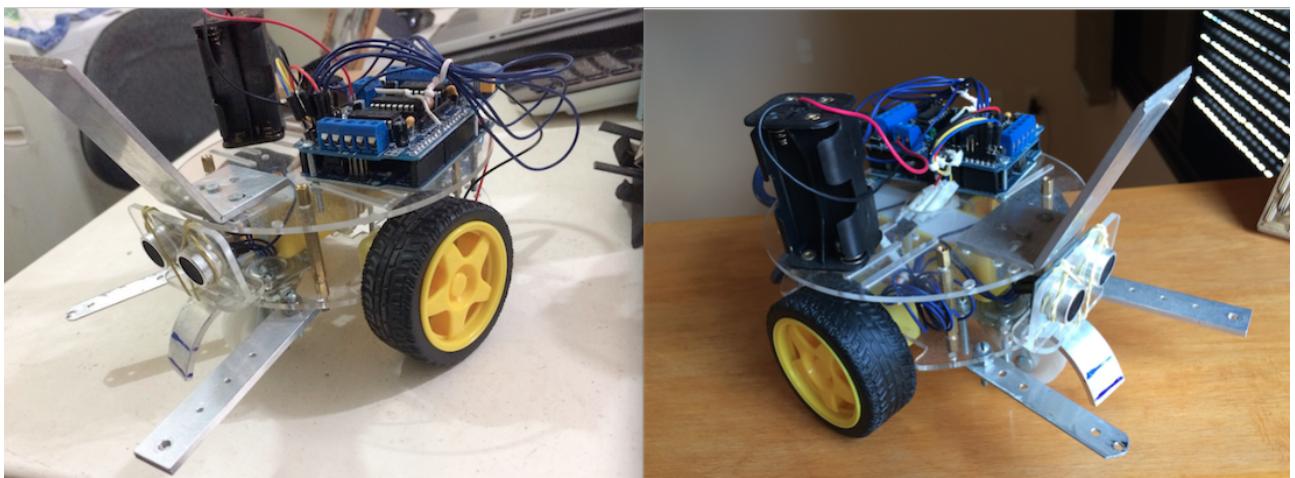
Para o desenvolvimento, o ambiente de atuação do sistema é um ambiente controlado que foi definido para facilitar o processo de visão computacional. Este ambiente está descrito no Capítulo 7.

6.1 CAMADA INFERIOR

A camada inferior do projeto é responsável por tratar da locomoção do robô, realizar sensoriamento do ambiente e se comunicar com a camada intermediária (recebendo instruções e enviando informação).

O robô usado neste projeto é o *Meg-alie* (Figura 18) e foi produzido pelo autor. Esse possui uma placa *Arduino UNO*, dois motores de corrente contínua, um driver para os motores, um sensor de distância ultrassônico, uma bateria portátil e um suporte para pilhas – além de um módulo de comunicação *bluetooth*. É um robô não-holonômico⁴ que tem um chassi de acrílico circular com 15 centímetros de diâmetro e pesa, aproximadamente, 500g.

Figura 18 – *Meg-alie*.



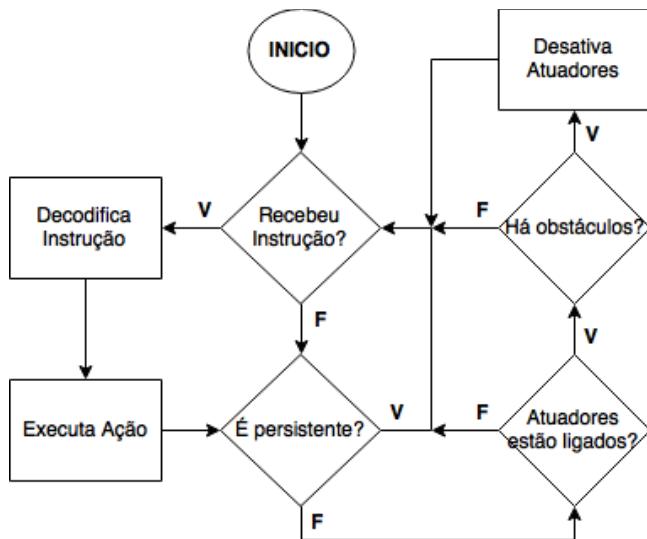
Fonte: elaborada pelo autor.

O *Meg-alie* foi programado para, a cada ciclo, verificar se recebeu alguma instrução. Caso tenha recebido, ela é decodificada. Após a decodificação, o robô executa a tarefa associada à

⁴ Vide capítulo 5.2.

instrução. O robô, também, realiza a cada ciclo uma verificação para saber se pode manter os atuadores funcionando. Este processo, no caso do *Meg-alie*, é somente verificar se há algum obstáculo próximo através do sensor de distância e, se houver, os motores são desativados. Caso seja necessário que o robô continue com os atuadores ligados mesmo que haja um objeto próximo (caso ele precise arrastar o objeto, por exemplo), é possível ativar seu estado de **persistência**: o robô persistente ignora a verificação e mantém os atuadores ligados. A Figura 19 ilustra o ciclo executado pela camada inferior.

Figura 19 – Ciclo da camada inferior.



Fonte: elaborada pelo autor.

A instrução é recebida pelo robô através de uma estrutura de dados de 1 byte. Os três primeiros bits definem o código da instrução a ser executada e os últimos cinco definem os parâmetros da instrução. Desta forma, a decodificação da instrução se dá por:

$$\text{Instrução} \leftarrow \text{BBBBBBBB} \wedge 11100000 ,$$

em que B representa cada bit estrutura recebida.

A Tabela 3 fornece as instruções, seus nomes e detalhes recebidas pela camada inferior. Com relação aos parâmetros: M define o motor, onde M = 1 é o motor esquerdo; D define a direção, onde D = 1 é para trás; V define a velocidade do motor, onde V = 6 representa a velocidade máxima do motor; T define o tempo de rotação do robô (em 10 milissegundos); e P define se o robô é ou não persistente sendo que, se P = 1, o robô se torna persistente.

Tabela 3 – Instruções da camada inferior.

BINÁRIO	NOME	DESCRIÇÃO
001MDVVV	Ligar motor	Liga somente um motor, escolhendo a direção e a velocidade de rotação.
010XDVVV	Ligar todos os motores	Liga todos os motores, escolhendo a direção e a velocidade
011TTTTT	Rotacionar para direita	Aciona os motores de forma que o robô rotacione no sentido horário
100TTTTT	Rotacionar para esquerda	Aciona os motores de forma que o robô rotacione no sentido anti-horário
101XXXXX	Enviar distância	Envia a distância medida pelo sensor de distância para a camada intermediária
110XXXXP	Troca persistência	Altera o estado de persistência do robô
111XXXXX	Parar motores	Para completamente ambos os motores

Fonte: elaborada pelo autor.

6.2 CAMADA INTERMEDIÁRIA

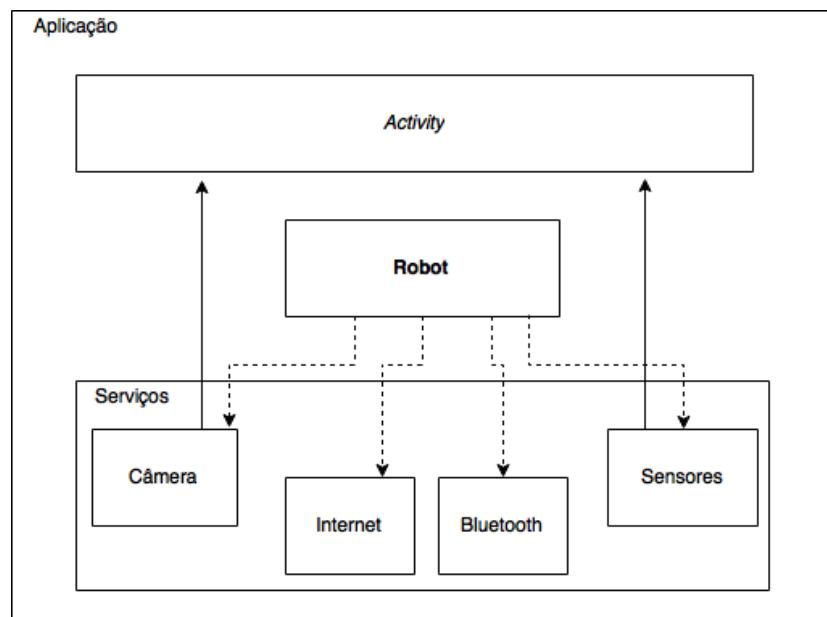
A camada intermediária é a camada responsável por obter a imagem do ambiente, enviar instruções para a camada inferior e fazer a transferência de dados com a camada superior. Ela, também, é quem executa o algoritmo de busca e coleta de objetos no ambiente. Nesta camada, o dispositivo de *software* programado foi um *smartphone* Samsung Galaxy Y, que tem o *Android* 2.3.6 instalado. Foram usadas a linguagem de programação *Java* e a IDE *Android Studio* para implementar o dispositivo.

Na Figura 20 foi definida a estrutura da aplicação desenvolvida. As setas cheias indicam dependência total de uma classe para a outra. Assim, a câmera e os sensores dependem da *activity* para existirem. As setas pontilhadas indicam dependência parcial de uma classe para a outra. Neste caso, a classe **Robot**, pode existir sem nenhum dos serviços. As classes de câmera, internet, sensores e bluetooth fazem parte de uma interface de serviços, a qual é responsável por padronizar o uso dos recursos do *smartphone*. A classe **Robot** faz uso desses serviços implementando métodos que os iniciam e os interrompem.

A interface da aplicação (Figura 21) foi feita para aproveitar da modularidade da implementação. A interface possui 3 abas: uma de configurações, outra de controle manual e outra de controle automático.

A aba de configurações permite a personalização das configurações do robô. Com ela, é possível alterar o endereço *bluetooth* do robô a ser usado, o endereço IP e as portas de comunicação de rede do computador da camada superior e ainda é possível habilitar o uso dos sensores do celular ou a câmera.

Figura 20 – Estrutura da aplicação *Android*.



Fonte: elaborada pelo autor.

O controle manual permite que o robô seja controlado como um brinquedo de controle remoto, em que o celular é o controle. Este recurso permitiu o teste da parte mecânica e elétrica do robô.

Figura 21 – Interface da aplicação.



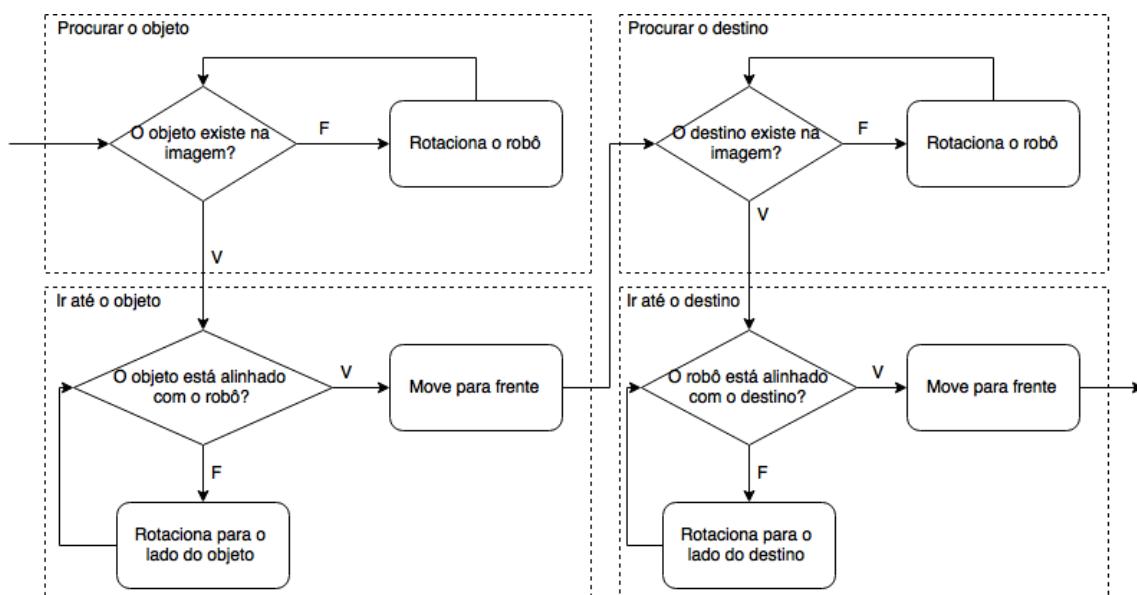
Fonte: elaborada pelo autor.

Por fim, o controle automático executa o algoritmo de busca e coleta de objetos com os recursos disponíveis pelo celular e habilitados na aba de configurações. A aba de controle automático exibe os dados obtidos pela câmera, caso esta esteja habilitada. Neste modo, também, o dispositivo se comunica com a camada superior para a obtenção de informação relevante da imagem da câmera. A comunicação, tal como a da camada inferior com a intermediária, se dá pelo envio de 1 byte (código) pela camada intermediária através da rede. Foram implementadas quatro mensagens:

- Quando a camada superior recebe o código 0, termina a comunicação entre as duas camadas;
 - Quando recebe o código 1, ela se prepara para receber uma imagem da camada intermediária;
 - Os códigos 2 e 3 fazem com que a camada superior extraia da imagem a posição de uma objetivo na imagem. Se o código for 3, o objetivo é o objeto a ser empurrado pelo robô. Se o código é 2, o objetivo é o destino do objeto.

O algoritmo de busca implementado consiste, basicamente, em procurar o objeto, ir até o objeto, procurar o destino e empurrar o objeto até o destino. Entretanto, como as instruções da camada inferior e intermediária são restritas a pequenos movimentos, estes passos precisaram ser decompostos em passos menores. No fim, o processo se segue como no fluxo da Figura 22. O tempo de rotação é variável e é alterado pelo robô conforme for necessário ao longo da busca.

Figura 22 – Fluxo do processo de busca e coleta de objetos



Fonte: elaborada pelo autor.

6.3 Camada superior

A camada superior trabalha com problemas mais complexos e que exigem maior poder de processamento. Ela é responsável por responder às solicitações da camada intermediária, processar as imagens por esta enviadas e extrair informações relevantes delas.

Neste camada, a visão computacional foi implementada utilizando a linguagem de programação C++ e a biblioteca *OpenCV*. Esta camada, basicamente, inicia criando um servidor que terá como cliente a camada inferior e assim que o cliente estiver conectado ela inicia a leitura de instruções (descritas no capítulo anterior).

O sistema de visão implementado foi baseado no descrito no capítulo 5.3 por Brooks, Grimson e Lorigo (1997). Entretanto, a técnica descrita tem como finalidade evitar obstáculos, não identificá-los. Além disso, a proposta deles seria de trabalhar na maior quantidade de ambientes o possível. Como a proposta deste projeto é de identificar um objeto e um local do ambiente e como o ambiente de trabalho é controlado o método foi readaptado para esta situação.

As adaptações feitas do método para este projeto são:

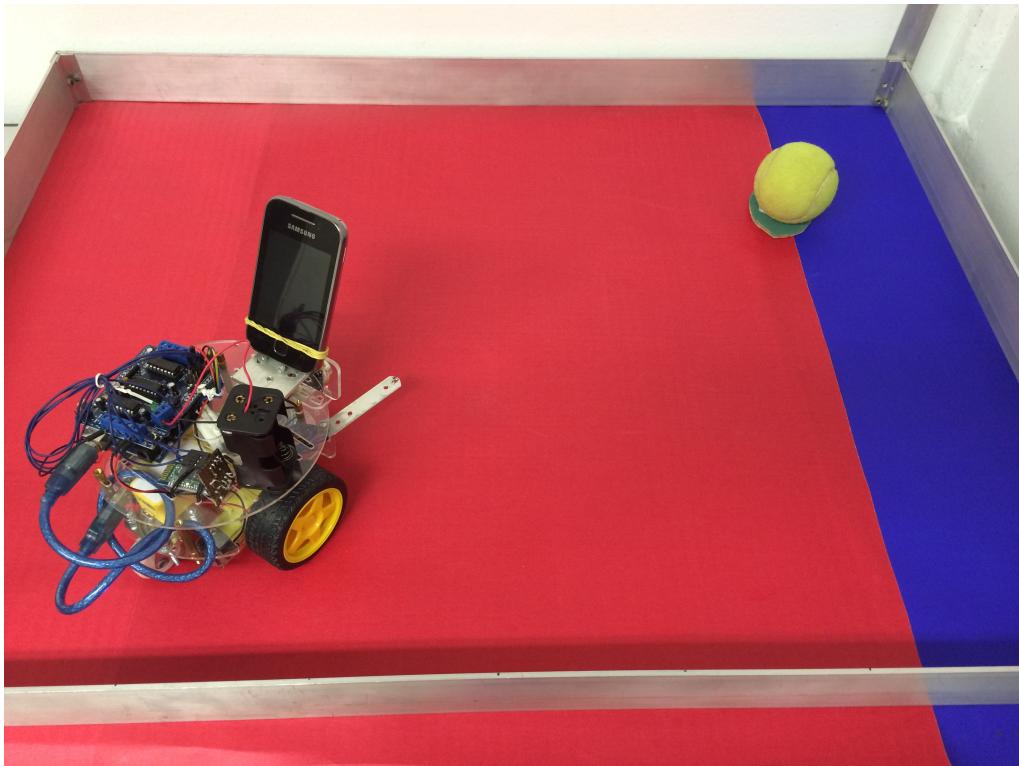
- Como o ambiente possui cores contrastantes, foi usado somente o módulo RGB para comparação;
- Originalmente, a comparação seria feita utilizando dois quadros. Entretanto, como o objetivo deste projeto é identificar objetos(não evitá-los), foi usado somente um quadro e a área deste era comparado com um valor obtido experimentalmente do objeto desejado;
- A comparação, neste caso, dispensa o módulo, podendo aceitar valores negativos ou positivos. Assim, a comparação é dada por $\text{comparação} = \sum_v (h(v) - s(v))$ em que h é o histograma do quadro, s é um histograma previamente obtido e v é o valor da intensidade da distribuição escolhido;
- Com a remoção do módulo para a comparação das áreas, quando resultado da comparação é negativo não existe o objeto no quadro atual. Quando a comparação é igual ou maior que zero, o objeto existe no quadro.

Com estas adaptações, foi possível identificar o objeto desejado e a área de depósito através da visão computacional.

7. TESTES E RESULTADOS

Inicialmente o ambiente de atuação desejado para o sistema proposto era um campo de tênis. Entretanto, por conta da limitação do *hardware* e da complexidade em lidar com um ambiente real, foi feita uma aproximação para um campo menor, com cores contrastantes, facilitando, assim, a identificação do objeto no ambiente. Também foi definido que o objeto a ser identificado seria uma bola de tênis, visto que é um objeto que pertence naturalmente ao ambiente inicial e que é mais fácil de ser identificado por causa de sua cor. Este ambiente foi planejado para ser parecido com um canto lateral de uma quadra de piso duro⁵.

Figura 23 – Ambiente de atuação do sistema.



Fonte: elaborada pelo autor.

O ambiente controlado possuía 68x60 centímetros e limitadores de alumínio para que o robô não saísse dele. As cores vermelho e azul foram definidas pois são facilmente identificáveis no processo de visão computacional.

No teste, a bola de tênis era colocada dentro da área vermelha e longe do campo de visão do robô. Ambos eram posicionados longe das bordas. Assim que o sistema começava a funcionar, o robô precisava rotacionar diversas vezes até encontrar a bola. Além de rotacionar, ele também se

⁵ As quadras de tênis de piso duro são quadras que tem o piso feito de cimento ou asfalto.

movia um pouco para frente para o caso da bola estar muito longe para ser identificada pela visão. Quando a bola era encontrada, o robô ia até ela. Depois disso, este precisava procurar a região azul da quadra e, logo que fosse encontrada, empurrava a bola até lá. Por fim, ele se distanciava da bola movendo para trás.

Esses testes foram executados diversas vezes e, em várias vezes, o robô conseguia alcançar seu objetivo. Entretanto, muitas vezes os motores emperravam e com muita frequência era necessário parar os experimentos para recarregar as baterias do robô. Além disso, Brooks, Grimson e Lorigo (1997) citam em seu trabalho que a identificação por meio da análise de cores pode ser prejudicada em ambientes que possuem peças metálicas ou que refletem, de alguma forma, as cores dos objetos ao redor. Esta observação mostrou-se verdadeira visto que em alguns momentos o robô identificou a bola de tênis ou o objetivo nas bordas do ambiente, que são metálicas. Também, durante o processo de identificação da bola, outra situação que causava confusão nos resultados era a de quando o robô se aproximava muito das bordas e o campo de visão da câmera do robô captava imagens do que estava externo ao ambiente de atuação.

É possível ver o robô executando com sucesso sua tarefa no endereço <https://youtu.be/mgKfLhOAlu8>. Já no endereço <https://youtu.be/v7vVQJlf7yc> há um vídeo exemplificando o caso de falha, onde ele identifica a borda metálica do ambiente ao invés do objetivo.

8. CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS

Neste projeto foi planejado, construído e implementado um sistema robótico modular capaz de procurar e identificar uma bola de tênis usando visão computacional e de transportá-la até um local determinado. O sistema é dividido em três camadas que possuem aplicações específicas e que se complementam para alcançar o objetivo.

O ambiente de atuação do robô, neste trabalho, teve muitas restrições. Pretende-se, em trabalhos futuros, reduzir as restrições no ambiente, aumentar a gama de objetos ou aplicar técnicas de inteligência artificial para contextualização do meio.

Além disso, como a arquitetura de *software* do robô é modular, é possível reaproveitar cada camada em diferentes projetos ou diferentes sistemas com um mínimo de alteração no código-fonte. Desta forma, nos próximos projetos, há a possibilidade de aproveitar um sistema de visão computacional de outros trabalhos ou, ainda, reutilizar o robô com outros equipamentos nas outras camadas.

Também, durante o desenvolvimento, a maior dificuldade estava em manter a comunicação entre as três camadas estável e de descobrir a origem de *bugs* no sistema, já que o erro poderia estar em qualquer uma delas.

Por fim, todo o sistema e sua documentação foram disponibilizados no *github* em <https://github.com/cabraile/Modular-Mobile-Robot> para facilitar a reprodução do projeto.

REFERÊNCIAS

- ARDUFC. Diferença entre as placas arduino.** ArdUFC, 2012. Disponível em: <<http://ardufc.blogspot.com.br/2012/10/modelos-do-arduino.html>>. Acesso em: 11 de nov. 2015.
- BRADSKI, G. R.; KAEHLER, A. Learning OpenCV.** 1 ed. Sebastopol: O'Reilly Media, 2008.
- BROOKS, R.; GRIMSON, W.; LORIGO, L. M.** Visually-Guided Obstacle Avoidance in Unstructured Enviroments. In: Proceedings of the 1997 IEEE/RSJ International Conference on Intelligent Robots and Systems, 10., 1997, Grenoble. **Resumos...** [s.l.]: [s.n.], 1997, p. 373-379.
- CANNY, J.** A computational approach to edge detection. **IEEE transactions on pattern analysis and machine intelligence**, Washington, v. 8, n. 6, p. 679-698, jun. 1986.
- FERREIRA, G. A. N. Abordagens de utilização de arquiteturas middleware em aplicações robóticas embarcadas.** Exacta, São Paulo, v. 4, n. 1, p. 149-157, jan./jun. 2006.
- HANBURY, A.** The Taming of the Hue, Saturation and Brightness Colour Space. In: Proceedings of the 7th CVWW, 7., 2002, Bad Aussee. **Resumos...** [s.l.]: [s.n.], 2002, 234-243.
- HANZRATECH.** Thresholding Hue Component. Hanzratech, 7 fev. 2015. Disponível em <<http://hanzratech.in/2015/02/07/caveat-thresholding-hue-component.html>>. Acesso em 20 de jan. 2016.
- INTERNATIONAL DATA CORPORATION.** **Smartphone OS Market Share:** 2015 Q2. [s.l.], 2015. Disponível em: <<http://www.idc.com/prodserv/smartphone-os-market-share.jsp>> Acesso em 15 de dez. 2015.
- LABORATÓRIO DE GARAGEM.** Sensor de Proximidade Infravermelho de curto alcance Sharp GP2Y0A21YK. Laboratório de garagem, [s. d.]. Disponível em <<http://www.labdegaragem.org/loja/sensor-de-proximidade-infravermelho-sharp-gp2y0a21yk.html>> Acesso em: 11 de nov. 2015.
- LEONARD, J. J.; DURRANT-WHYTE, H. F.** Mobile robot localization by tracking geometric beacons. **IEEE Transactions on Robotics and Automation**, [s.l.], v. 7, n. 3, p. 376-382, jun. 1991.
- MATARIĆ, M.J. The robotics primer.** Cambridge: MIT Press, 2007.
- MURPHY, R. R. An Introduction to AI robotics.** Cambridge: MIT Press, 2000.

- NASCIMENTO, U. P. **Projeto de desenvolvimento de um robô expositivo.** Brasília: [s.n.], 2013.
- RANIERI, C. **Sistema para navegação de robôs móveis e interação humano-robô baseada em comandos de voz.** Bauru, [s.n.], 2013.
- SHWARZ, R. et al. **The Android developer's cookbook: building applications with Android SDK.** 2. ed. Crawfordsville: Addison-Wesley Professional, 2013.
- SIEGWART, R.; NOURBAKHSH, I.; SCARAMUZZA, D. **Introduction to Autonomous Mobile Robots.** [s.l.]: MIT Press, 2004.
- SILVA, M. O. **Campos potenciais modificados aplicados ao controle de múltiplos robôs.** 2011. Dissertação (Mestrado em Ciências de Computação e Matemática Computacional) - Instituto de Ciências Matemáticas e de Computação, University of São Paulo, São Carlos, 2011. Disponível em: <<http://www.teses.usp.br/teses/disponiveis/55/55134/tde-23112011-090055/>>. Acesso em: 2015-08-16.
- SZELISKI, R. **Computer Vision: Algorithms and Applications.** New York: Springer-Verlag, 2010.
- TANENBAUM, A. S.; WETHERALL, D. J. **COMPUTER NETWORKS.** 5. ed. Amsterdam: Prentice Hall Professional, 2002.
- [s.n.]. **Servo Motor.** [s.l], [s.d] Disponível em: <<http://www.feis.unesp.br/Home/departamentos/engenhariaeletrica/aula-4---servo-motor-13-03-2013-final.pdf>>. Acesso em 30 ago. 2015.