UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Danilo Vieira França

Navegação Autônoma em Robôs de Baixo Custo com *Smartphone* como Sensor e Cérebro

Uberlândia, Brasil

2025

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Danilo Vieira França

Navegação Autônoma em Robôs de Baixo Custo com *Smartphone* como Sensor e Cérebro

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, como parte dos requisitos exigidos para a obtenção título de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Jefferson Rodrigo de Souza

Universidade Federal de Uberlândia – UFU

Faculdade de Ciência da Computação

Bacharelado em Ciência da Computação

Uberlândia, Brasil

2025

Resumo

Este trabalho apresenta uma plataforma robótica de baixo custo, baseada em Arduino, que integra um *smartphone* Android como único sensor visual e unidade de processamento, para detectar e perseguir um objeto através de Visão Computacional. O robô, controlado por Arduino, executa navegação a partir de comandos recebidos via Bluetooth, gerados pelo aplicativo Android que emprega OpenCV e detecção de objetos com base na cor (Blob Detection). 40 execuções em dois cenários padronizados compararam a solução a um robô seguidor de linha tradicional, equipado com sensores infravermelhos, revelando desempenho equivalente, porém com muito mais flexibilidade operacional. Adicionalmente, são realizados novos experimentos com um segundo *smartphone*, mais moderno e, por consequência, com maior poder computacional, o que possibilitou a exploração dessas novas características para obter resultados mais precisos. Isso inclui o teste de estimativas de distâncias, onde foram realizadas estimativas para distâncias entre 5 e 210 centímetros e calculados os erros para a maior e menor distância estimadas. Com base nisso, foram elaboradas 4 abordagens com o objetivo de reduzir esses erros. Este trabalho apresenta como contribuição a demonstração de um *smartphone* combinado a um kit robótico, substituindo sensores ópticos dedicados e navegação autônoma usando bibliotecas *open-source* no ambiente móvel.

Palavras-chave: Visão Computacional, Robótica de Baixo Custo, *Smartphone* como Sensor, Detecção de Objetos, Navegação Autônoma.

Lista de ilustrações

Lista de tabelas

Sumário

1 Introdução

A robótica móvel, impulsionada pelo aprimoramento contínuo da tecnologia de sensores e da capacidade computacional, enfrenta desafios significativos que permanecem em aberto para exploração. Câmeras de alta resolução tornaram-se "onipresentes"graças aos 4,69 bilhões de *smartphones* em uso em 2025, quase cinco vezes mais que em 2015, e a projeção é chegar a 5,83 bilhões já em 2028 (**Tabela 1**).

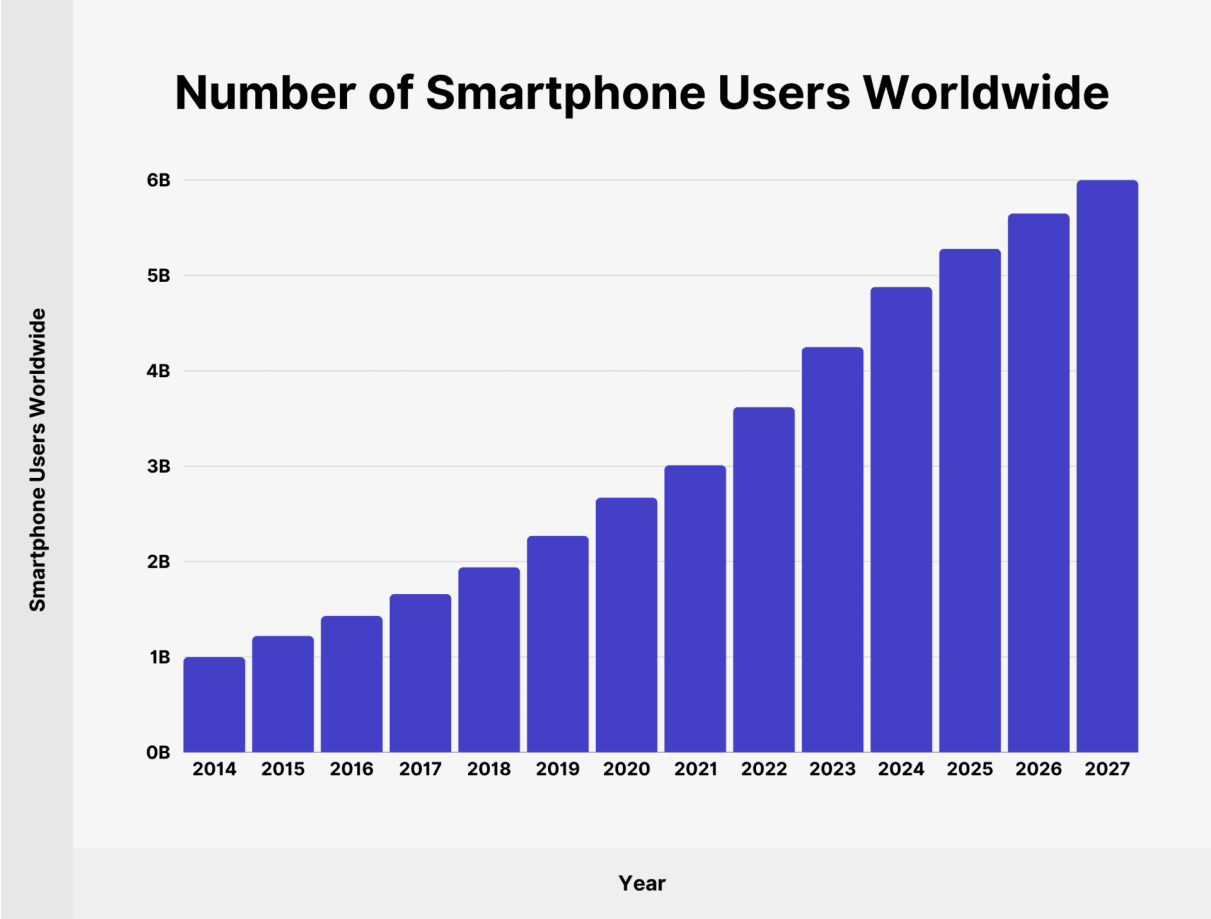
Apesar de existirem módulos ópticos mais baratos, como a ESP32-CAM (entre 6 e 15 dólares) (MAKERFOCUS, 2025) e a Raspberry Pi High Quality Camera (em torno de 55 dólares) (RASPBERRY PI FOUNDATION, 2025), esses sensores enfrentam limitações significativas em termos de resolução, precisão na estimativa de distância e capacidade de processamento embarcado, além de exigirem montagens complexas e componentes adicionais, como lentes intercambiáveis e placas dedicadas (ex. Raspberry Pi). Por outro lado, sensores ópticos especializados, como câmeras industriais Basler (por volta de 500 dólares) (BASLER AG, 2025), *depth-cameras* Intel RealSense D435 (US$ 314) (INTEL CORPORATION, 2025) ou LiDARs como o RPLidar A1 (por volta de US$ 120) (SLAMTEC, 2025), facilmente excedem o orçamento disponível quando multiplicados por vários kits para uso em sala de aula e projetos de extensão, limitando sua ampla adoção em ambientes educacionais (STEIN et al., 2025).

Nesse cenário, *smartphones* usados surgem como alternativa atraente, já que oferecem resolução elevada, poder computacional integrado e dispensam qualquer complexidade adicional de montagem, além da alta disponibilidade de dispositivos, permitindo uma fácil reprodutibilidade e um acesso mais democrático à robótica móvel e visão computacional em contextos de baixo custo. Paralelamente, o poder computacional embarcado nos telefones passou a rivalizar com computadores usados em robótica. Isso somado faz com que um *smartphone* se torne plataforma sensorial completa, capaz de substituir vários módulos externos quando acoplados a um chassi robótico simples.

Este trabalho tem como objetivo demonstrar, por meio de avaliação quantitativa, que um *smartphone* Android reutilizado, executando OpenCV, pode desempenhar simultaneamente os papéis de sensor visual e unidade de processamento, guiando um robô Arduino, através de detecção de objeto, em tarefas de navegação autônoma. Essa abordagem será comparada com uma configuração tradicional de robótica: um robô seguidor de linha. Além disso, são exploradas as possibilidades desse modelo.

A proposta busca demonstrar a eficácia dessa configuração em uma tarefa crucial da robótica móvel: detecção e perseguição de objetos. Além disso, são exploradas as possibilidades desse modelo.

Tabela 1 – Usuários globais de *smartphones* ao longo dos anos, extraído de (Backlinko Team, 2025)



O objetivo é desenvolver uma plataforma eficiente e de baixo custo, aproveitando as capacidades das câmeras e do processamento dos *smartphones*. Essa abordagem não apenas democratiza o acesso à tecnologia, mas também promove oportunidades educacionais, especialmente em ambientes acadêmicos, estimulando o avanço do conhecimento na área.

Este trabalho evidencia a integração de um *smartphone* diretamente a um kit robótico, eliminando a dependência de sensores ópticos especializados. Ao explorar o poder computacional do telefone embarcado e bibliotecas *open-source*, demonstra-se um caminho acessível para a Visão Computacional em ambientes educacionais e pesquisa com orçamento limitado.

Na próxima seção, será apresentada a Fundamentação Teórica, que descreve o contexto teórico que embasa o trabalho; na sequência vêm os Trabalhos Relacionados que abordam a robótica móvel sob diferentes perspectivas, onde essa temática é abordada por outros ângulos ou outras soluções são apresentadas, fundamentando a necessidade deste trabalho. A metodologia empregada, detalhada na seção correspondente, descreverá a arquitetura proposta e como a integração com o *smartphone* foi realizada. Os experimentos,

discutidos na seção de Resultados, validam a proposta. Finalmente, a seção de Conclusão discutirá as implicações das abordagens utilizadas, a contribuição do trabalho e oferecerá sugestões para desafios futuros.

2 Fundamentação Teórica

O Android é um Sistema Operacional baseado no kernel do Linux desenvolvido pela Google e outras companhias que formam a Open Handset Alliance (KING, 2016). Segundo (PEREIRA; SILVA, 2009), o Android foi construído com a intenção de permitir que os desenvolvedores projetem aplicações que façam uso dos recursos de um dispositivo.

Essas aplicações, conforme explica (PEREIRA; SILVA, 2009), possuem um usuário Linux com seus próprios diretórios, qualquer tentativa de acesso às informações de outros aplicativos ou recursos do dispositivo precisa ser autorizada explicitamente pelo usuário, isso dá ao sistema operacional uma alta segurança.

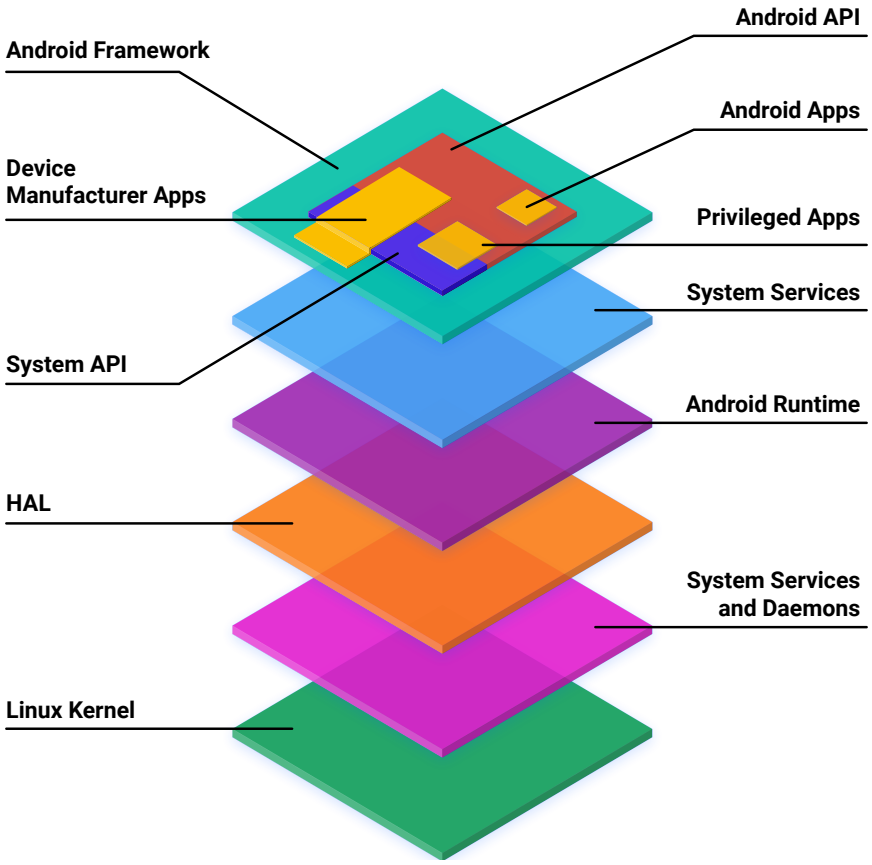


Figura 1 – Arquitetura Android, extraído de (Android Developers, 2024).

O aplicativo desenvolvido é executado na camada de aplicação, no entanto, também faz uso de recursos das camadas inferiores, como acesso à câmera, Bluetooth e armazenamento.

A detecção do objeto é realizada através de Processamento Digital de Imagens, para isso foi utilizado o OpenCV. (GONZALEZ; WOODS, 2002) explica que não há um consenso entre os autores sobre onde começa e termina o conceito de processamento digi-

tal. Uma das delimitações consideradas é a de três tipos de processos computadorizados: *low-level*, que consiste em operações mais primitivas como redução de ruído, aprimoramento de contraste e nitidez de imagem; *mid-level*, que seria a extração de atributos de uma imagem através de segmentação, descrição e classificação de objetos; e *high-level* que envolve "atribuir significado"a um conjunto de objetos reconhecidos. Bibliotecas como o OpenCV oferecem implementações otimizadas desses estágios em C/C++ e Java, inclusive para Android.

OpenCV é uma biblioteca de código aberto de visão computacional e aprendizado de máquina (BRADSKI; KAEHLER, 2008). Visão Computacional é um campo muito vasto, (BRADSKI; KAEHLER, 2008) define como sendo a transformação de dados de uma imagem ou uma sequência de imagens de um vídeo em uma decisão ou uma nova representação; essas transformações são feitas para se atingir algum objetivo específico.

Arduino é uma plataforma eletrônica de baixo custo *open source* de fácil aprendizagem, compatível com módulos periféricos e sistemas operacionais (ARDUINO, 2021). A maior vantagem do Arduino em relação a outras plataformas de desenvolvimento de microcontroladores é a sua facilidade de utilização, o que permite que pessoas que não sejam de áreas técnicas possam aprender o básico e criar seus próprios projetos em um período relativamente curto (MCROBERTS, 2018).

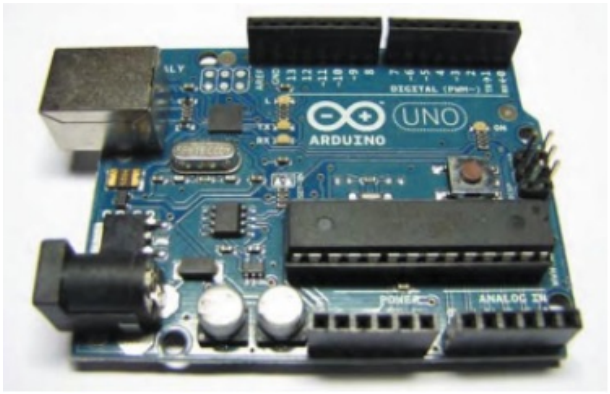


Figura 2 – Placa Arduino UNO utilizada no trabalho, (MCROBERTS, 2011).

Em robótica móvel, o ciclo de navegação é decomposto em quatro funções interdependentes: percepção, que obtém a situação do ambiente a partir dos sensores. A localização, responsável por estimar a pose do robô, onde ele está no espaço e para que lado ele está voltado. O planejamento, que gera uma trajetória viável até o objetivo. E por fim, o controle, converte a trajetória em comandos de motor (THRUN; BURGARD; FOX, 2005). O aplicativo usa o OpenCV para fazer a detecção do objeto e envia a decisão de navegação para o robô via Bluetooth; a navegação com base na detecção do objeto é feita conforme mostrado por (DESOUZA; KAK, 2002).

Neste trabalho, foi adotado o modelo de câmera *pin-hole*, no qual a lente real é aproximada por um ponto único e o sensor por um plano da imagem situado a uma

distância f (distância focal). Nessa configuração, a projeção de um ponto (X, Y, Z) no espaço para coordenadas de imagem (x, y) é dada por

$$(x, y) = (fX/Z, fY/Z),$$

conforme Eq. (6.1) de (HARTLEY; ZISSERMAN, 2004). Para um objeto de altura real H paralelo ao eixo Y, a diferença h de *pixels* entre o topo e a base resulta em

$$h = \frac{fH}{Z} \Rightarrow Z = \frac{fH}{h},$$

permitindo estimar a distância Z em tempo real a partir de uma única câmera, atendendo ao requisito de baixo custo.

A Detecção de Objetos corresponde à etapa *high-level* da Visão Computacional que localiza e classifica instâncias de interesse em cada quadro da câmera. Os algoritmos podem ser organizados em três famílias, cada qual com compromisso distinto entre robustez e demanda computacional.

Nos algoritmos baseados em segmentação por cor ou intensidade, métodos como *Blob Detection* e *Background Subtraction* realizam apenas operações de espaço de cor, limiarização e morfologia. São ideais quando existe forte contraste entre objeto e fundo, mas degradam sob variação de iluminação ou oclusões.

Detectores baseados em descritores fixos como por exemplo, Haar-cascade (VIOLA; JONES, 2001) ou HOG combinado a SVM (DALAL; TRIGGS, 2005) aplicam filtros convolucionais pré-definidos seguidos de classificadores estatísticos. Oferecem maior robustez a iluminação/oclusões e suportam múltiplas classes, mas exigem um arquivo de modelo pré-treinado e esforço de coleta de dados.

Arquiteturas de rede neural convolucional, como YOLO (REDMON et al., 2016), SSD ou Faster R-CNN (LIU et al., 2019), alcançam precisão de ponta em *benchmarks*, detectam dezenas de classes por imagem e escalam bem em ambientes não controlados. Aplicações educacionais frequentemente delegam a inferência para um servidor remoto ou simplificam o modelo (Tiny-YOLO).

Essas categorias apresentam uma graduação da relação entre rapidez de implementação e robustez. O *Blob Detection* opera na extremidade "rápido, mas especializado" do espectro. Ele sacrifica generalidade (precisa de fundo razoavelmente limpo e cor distinta) em favor de performance em hardware modesto, o que o torna adequado para tarefas de navegação controlada como a avaliada neste Trabalho.

3 Trabalhos Relacionados

(OROS; JEFFREY, 2013) mostraram a amplitude de projetos que podem ser desenvolvidos usando um robô com *smartphone* e o sistema operacional Android. Foram construídos dois robôs altamente modulares e de baixo custo para serem utilizados em dois projetos diferentes, um realizado por alunos de engenharia e outro por um grupo de pesquisa em neurociência computacional. Os robôs foram capazes de concluir as suas tarefas com bom desempenho, mostrando que são uma solução viável para projetos tanto para estudantes quanto para pesquisadores. Os robôs construídos no artigo possuem uma arquitetura semelhante à usada neste trabalho, destacando a adaptabilidade dessa abordagem em contextos educacionais e de pesquisa.

(MENDOZA, 2014) propôs a movimentação de um robô autônomo usando somente visão computacional. Foram utilizados três módulos, um primeiro responsável pela captura de imagens do ambiente, tendo uma visão por cima, e calcular a rota entre dois pontos, o segundo módulo, que é um *smartphone* Android, recebe as imagens do primeiro módulo, detecta objetos de interesse (obstáculos, posição inicial, objetivo, limites do ambiente) através de operações de visão computacional e envia essas informações ao primeiro módulo; por fim, o terceiro módulo, que é o próprio robô, apenas recebe do segundo módulo as instruções da movimentação a ser realizada. Esse sistema obteve resultados satisfatórios, se mostrando capaz de guiar o robô usando apenas visão computacional. O robô desse trabalho também se propõe a se orientar pelo espaço apenas usando a câmera, porém não haverá essa divisão em módulos.

(FUSCO; ASSIS, 2015) aprimoram os *hardwares* e *softwares* de um robô autônomo que se movimenta com o auxílio de uma câmera de forma semelhante ao sistema humano. O sistema permite a proximidade com um sistema eficiente de sensoriamento. E (MORETTO; MELLO, 2020) desenvolveram um algoritmo que permitiu a um robô se localizar e navegar de forma autônoma em um ambiente com pisos táteis usando visão computacional. Assim, apresenta uma plataforma robótica para localização e navegação de robôs.

(ALYMANI; KARAR; SHEHATA, 2024) apresentam um robô de baixo custo baseado em Raspberry Pi que rastreia alvos RGB por meio de uma câmera IP externa. Os autores comparam três variantes de controle: PID clássico, PID customizado com acionamento intermitente e PID adaptativo ajustado por lógica fuzzy. O PID (Proporcional–Integral–Derivativo) pode ser entendido como um algoritmo de ajuste automático: ele mede continuamente o erro (diferença entre o valor desejado e o real) e, com base em três ações (P corrige o erro atual, I corrige o erro acumulado e D freia a aproximação),

gera o comando que leva o sistema ao alvo. O PID regula a orientação e a distância do robô em relação ao objeto. Na proposta deste trabalho, o processamento é feito diretamente na câmera do *smartphone*, dispensando infraestrutura externa e explorando filtros estatísticos (média, desvio-padrão e IQR) para estabilizar as medições.

(HAN; RYU; NAM, 2024) propõem um sistema de rastreamento em tempo real para *smartphones* acoplados a uma base giratória motorizada. O método emprega duas variações do YOLO (uma série de algoritmos de detecção de objetos, baseados em redes neurais) batizadas como modo de precisão e modo de velocidade, detecta e acompanha alvos enquanto a base giratória regula o giro da câmera para manter o objeto no centro da imagem. Os testes mostraram que o modo rápido manteve alta taxa de quadros e o modo preciso alcançou boa precisão, demonstrando a viabilidade de aplicar técnicas de *deep learning* em dispositivos móveis com recursos limitados. O trabalho reforça a viabilidade de usar o *smartphone* como plataforma de visão e difere da presente pesquisa ao integrar técnicas de *deep learning* associadas à movimentação da câmera, enquanto neste trabalho, a câmera permanece fixa na base robótica. As estratégias de balancear precisão e velocidade indicam caminhos para futuras extensões com técnicas de deep learning de baixo custo em robôs autônomos.

(STEIN et al., 2025) Apresentam uma plataforma de simulação em nuvem que replica a API do conjunto físico RoboScape dentro do navegador, permitindo que alunos programem e testem robôs virtuais sem adquirir hardware. Toda a infraestrutura roda em servidores Rust e WebAssembly e utiliza NetsBlox como linguagem visual; a comunicação cliente-servidor ocorre via WebSocket/TCP, enquanto os robôs simulados trocam mensagens UDP com o back-end, idênticas às do hardware real. A principal contribuição é eliminar o custo de aquisição e manutenção do robô, facilitando o escalonamento em cursos grandes. A limitação, porém, é justamente a ausência de validação física, não se medem imprecisões reais de sensores nem o consumo de bateria.

(NANDHINI; AL-KHAN; KUMAR, 2025) Descrevem um carro robótico de baixo custo cujo movimento é teleoperado por Bluetooth a partir de um aplicativo Android. O celular atua apenas como interface de controle, a placa Arduino UNO recebe comandos e aciona os motores. O projeto demonstra resposta rápida e simplicidade de montagem, propondo usos em automação residencial e vigilância. Não há percepção autônoma, a navegação depende inteiramente do operador.

Percebe-se que esses trabalhos integram toda a percepção no *smartphone* embarcado, sem câmeras externas ou computadores de bordo. Este trabalho pretende abordar essa lacuna.

Tabela 2 – Comparação dos trabalhos relacionados e desta pesquisa

Autor/Ano	**Plataforma**	**Abordagem**	**Contribuição/Limitação**
(OROS; JEFFREY, 2013)	Robô + Android	Arquitetura modular de baixo custo	Êxito em sala de aula; não calcula distância até o alvo
(MENDOZA, 2014)	Robô, visão aérea + smartphone	Planejamento global por visão cenital	Boa navegação; exige infraestrutura de câmera superior
(FUSCO; ASSIS, 2015)	Robô autônomo + câmera USB	Visão monocular para alinhamento e aproximação	Melhora hardware/software; não discute custo nem métricas quantitativas
(MORETTO; MELLO, 2020)	Robô + câmera frontal	Seguimento de piso tátil por segmentação de cor	Solução específica para pisos táteis; baixa generalização
(ALYMANI; KARAR; SHEHATA, 2024)	Raspberry Pi + câmera IP	PID adaptativo para rastreamento RGB	Baixo custo; câmera externa fixa, robô não carrega o sensor
(HAN; RYU; NAM, 2024)	Smartphone em base giratória	YOLO-Lite com perfis “precisão” e “velocidade”	Exibe DL móvel; movimenta apenas a câmera, não o robô
(NANDHINI; AL-KHAN; KUMAR, 2025)	Arduino + smartphone	Teleoperação Bluetooth	Muito barato e responsivo; sem autonomia, alcance 10 m
(STEIN et al., 2025)	Navegador + nuvem	RoboScape Online (robô virtual)	Zero hardware, fácil escalar; sem validação física
Este trabalho	Arduino + smartphone	Blob Detection + filtros estatísticos para navegação	Sensor único embarcado, baixo custo, mede erro de distância; limitado a objetos de cor conhecida e iluminação controlada

4 Metodologia

A intenção da proposta é promover uma plataforma de baixo custo, que oferece funcionalidades de Visão Computacional. A plataforma robótica proposta é baseada em um kit robótico para Arduino, que consiste de uma estrutura de acrílico e metal sustentando quatro rodas motorizadas, e os sensores, periféricos e microcontrolador Arduino. Cada uma das quatro rodas do robô tem seu próprio motor, e cada motor pode ser ativado na polaridade invertida para girar no sentido contrário. Isto torna o robô holonômico, ou seja, capaz de rotacionar em torno de si mesmo (ativando os motores de cada lado em sentidos opostos), uma característica muito desejável para navegação em ambientes internos, que frequentemente têm pouco espaço disponível para manobras (ROMERO et al., 2014).

O Arduino é um microcontrolador de baixo custo, de interface simples e com um grande número de periféricos compatíveis. Apesar de suas vantagens, o Arduino é insuficiente para aplicações mais complexas, como as que envolvem processamento de imagens, devido às limitações de memória. Por isso, kits de robótica controlados por Arduino são comuns em aplicações que envolvem outros sensores.

Para solucionar tanto a necessidade de uma câmera quanto a capacidade de processamento, foi proposto o uso de um *smartphone* com sistema operacional Android em conjunto com o Arduino. Sendo assim, foi elaborado um apoio simples para o telefone celular, que é de fácil remoção, sem conexão cabeada com o restante do robô, conforme a (**Figura 3**). Isso satisfaz um dos requisitos importantes do sistema, que é o baixo custo, e permite que qualquer usuário possa utilizar o robô com seu próprio *smartphone*. A comunicação é feita através de um módulo Bluetooth instalado no Arduino, que se conecta ao celular.

Diferentemente de abordagens que mantêm a câmera externa ou dependem de computadores, a arquitetura proposta acopla o *smartphone* diretamente no chassi, executando todo o processo de percepção e tomada de decisão no dispositivo. Esta escolha reduz cabeamento, simplifica a montagem para iniciantes e torna a plataforma independente de infraestruturas ou computadores adicionais.

4.1 Algoritmo Android para navegação autônoma

O sistema proposto envolve dois softwares distintos: o aplicativo que executa no telefone Android (FRANçA, 2025c) e o software que executa no Arduino (FRANçA, 2025d). O aplicativo Android faz uso da biblioteca de visão computacional OpenCV, integrada

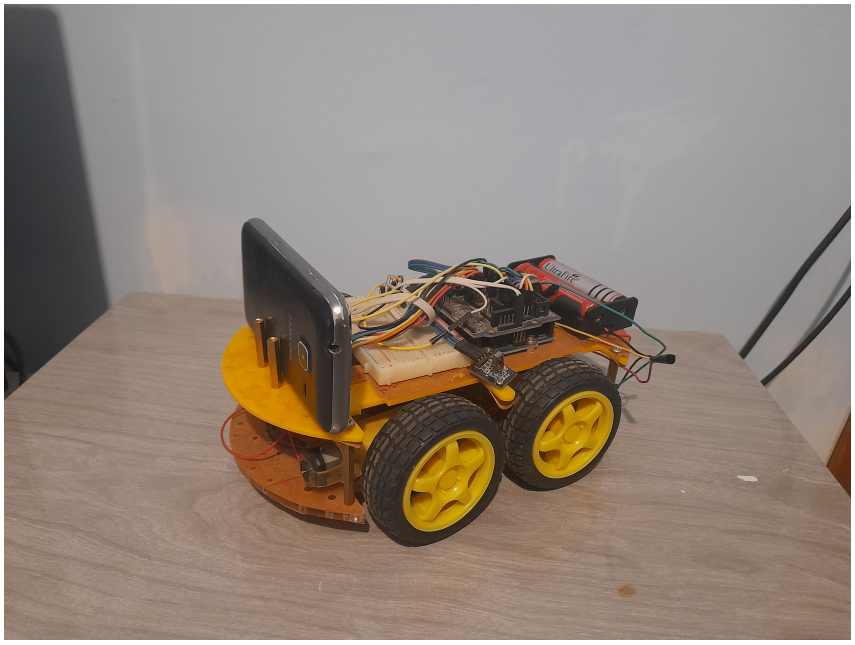


Figura 3 – Plataforma robótica móvel utilizada nos experimentos.

com o sistema Android através do SDK OpenCV4Android. O OpenCV oferece implementações de diversos métodos de extração de regiões, pontos de interesse ou características descritivas da imagem. Neste trabalho, foi utilizado o algoritmo *Blob detection* (detecção de região de interesse) que segmenta regiões de uma imagem onde *pixels* contíguos sejam semelhantes em valor de cor.

Ao iniciar o aplicativo desenvolvido, na tela inicial são apresentados os dispositivos *Bluetooth* pareados e, após escolher o dispositivo desejado, é conferido pelo aplicativo se há uma conexão serial entre os dispositivos. Para realizar uma conexão serial, utilizou-se o aplicativo *Serial Bluetooth Terminal* (MORICH, 2021). Em seguida, é feita a aplicação principal, que é capturar a imagem da câmera e aplicar o *Blob Detection* para a imagem capturada, a fim de detectar a característica desejada. Por fim, é enviado um comando ao robô via conexão *Bluetooth*.

4.1.1 Pipeline de visão no Android

A cada quadro de vídeo capturado, ocorre a execução do algoritmo de detecção de região de interesse, levando em consideração a cor do objeto alvo. A região é detectada pelo algoritmo *Blob detection*, e seu tamanho é medido em *pixels*. Como as dimensões reais do objeto alvo são conhecidas, a partir da altura e largura da região em *pixels*, é possível estimar a distância da câmera ao objeto detectado através de semelhança de triângulos. Pela posição horizontal da região detectada na imagem, podemos determinar se o robô está seguindo na direção do objeto. Se a câmera do *smartphone* perde o objeto alvo de vista, o robô deve rotacionar na direção em que o objeto alvo foi avistado pela última vez na imagem. Por fim, o robô pára quando chega a uma certa distância do objeto. Com

essa heurística de navegação, é decidido o comando de navegação a ser enviado ao robô.

Detalhamento dos passos da detecção do objeto:

1. Captura da imagem
2. Pré-processamento: Cada quadro sofre *pyr-down* duas vezes para acelerar o processamento; em seguida converte-se para HSV.
3. Segmentação cromática: Aplica-se $inRange()$ com limites HSV definidos em tempo de execução; o resultado é dilatado para preencher falhas.
4. Extração de contornos: $findContours()$ retorna um conjunto $C = \{c_i\}$.
5. Seleção do maior contorno: O *blob* c_b com maior altura em *pixels* (b_h) é assumido como o alvo.
6. Estimativa de distância:
$$d_k = \frac{f.h_{real}}{b_h},$$
onde f é a distância focal efetiva e h_{real} é a altura física do objeto (lata de alumínio de cor amarelada, 15,5 cm de altura).
7. Média das distâncias: Os últimos 15 valores de distâncias estimados são utilizados para calcular a média final d.
8. Cálculo de centroide: O centroide (x_c, y_c) de c_b é utilizado posteriormente para saber para que lado o objeto está.
9. Decisão de navegação: Avançar ("F") se $d \geq 21cm$; Parar ("S") se $d < 21cm$.
10. Caso nenhum contorno seja detectado, o robô gira para Esquerda ("L") ou Direita ("R") com base na última posição x_c.

O aplicativo envia o comando de navegação para o módulo Bluetooth instalado na placa Arduino. Esse comando é um *token* ASCII, o software do Arduino aguarda os comandos chegarem pela conexão Bluetooth. Os comandos são caracteres correspondentes a "seguir em frente", "girar para a esquerda", "girar para a direita"e "parar". O programa lê o comando, interpreta o *token*, aciona os motores e, em seguida, aguarda um curto intervalo para permitir que a câmera capture quadros nítidos antes do próximo comando. O aplicativo envia um único caractere seguido de ":"que age como delimitador, prevenindo *bytes* residuais no *buffer*. Ao receber um *token*, é executada a manobra correspondente e depois volta para um estado de espera. Cada movimento dura cerca de 200 ms e o tempo de espera dura entre 250 ms e 550 ms. Se nenhum *token* é lido, o comando padrão é de se manter parado. O tempo de espera foi definido para que a câmera consiga capturar

boas imagens, já que o contorno da região detectada oscila bastante, devido à própria movimentação do robô, que causa distorções na imagem obtida pela câmera com o robô se movendo para frente ou girando para os lados. Esse ciclo entre mover, parar e medir distâncias mitiga borrões que, durante movimento contínuo, fazem o contorno se distorcer. A cadência resultante é de aproximadamente 1 segundo por ação.

5 Resultados

5.1 Configurações

A metodologia proposta de detecção de objetos (OD) foi comparada com uma abordagem de robô seguidor de linha (LF). O seguidor de linha é uma aplicação que gera bons resultados, mas apresenta a limitação de depender de uma linha. Para o experimento com *line follower*, foram usados três sensores infravermelho, localizados na parte dianteira inferior do robô (**Figura 4**). Cada sensor possui um par emissor-receptor IR, o LED infravermelho emite radiação em um determinado comprimento de onda, parte dessa radiação é refletida pela superfície do chão e captada pelo fototransmissor adjacente. A intensidade do sinal refletido varia inversamente como o coeficiente de absorção da superfície, uma região clara reflete entre 60% e 70% do feixe, enquanto que a fita preta absorve mais de 90% (SEMICONDUCTORS, 2023), gerando um contraste típico no pino de saída analógica. Através das leituras dos sensores, o robô pode localizar a linha e ajustar sua direção de navegação. Sendo assim, o robô se moverá para frente de acordo com a leitura obtida pelo sensor do meio e corrigirá a trajetória conforme o que for obtido da leitura dos sensores da direita e esquerda, seguindo a linha no chão. Caso o robô saia dos "trilhos", o comando a ser executado será o de girar na direção correspondente ao último sensor que obteve a leitura da linha.

Foram elaborados dois circuitos diferentes para testar cada método. No primeiro cenário, a tarefa usada na comparação foi o percurso de um circuito em forma de pentágono com cerca de 60 centímetros de aresta (**Figura 5**), o segundo cenário consiste em um circuito em forma de retângulo (**Figura 6**). Na configuração de robô seguidor de linha (LF), o robô deve percorrer integralmente o trajeto delimitado, tocando sequencialmente os vértices, de 1 a 5 no Cenário 1 e de 1 a 4 no Cenário 2, sem qualquer intervenção humana, a não ser no início, onde o robô é movido para frente até o começo da linha. No modo detecção de objetos (OD), um único objeto-alvo é posicionado manualmente no *checkpoint* inicial (1, depois 2, e assim por diante). O objeto é transferido para o próximo *checkpoint* da sequência apenas quando o robô se aproxima do mesmo e pára. Ao mover o objeto, o robô continua a busca até completar todos os pontos previstos no cenário. Além disso, o robô é posicionado "de costas"para o objeto, dessa forma o robô começa o trajeto procurando pelo alvo em questão. Todos os experimentos foram feitos no mesmo ambiente, sob luz artificial. Os vídeos disponíveis em (FRANçA, 2025a) e (FRANçA, 2025b) ilustram como foi conduzido cada experimento.

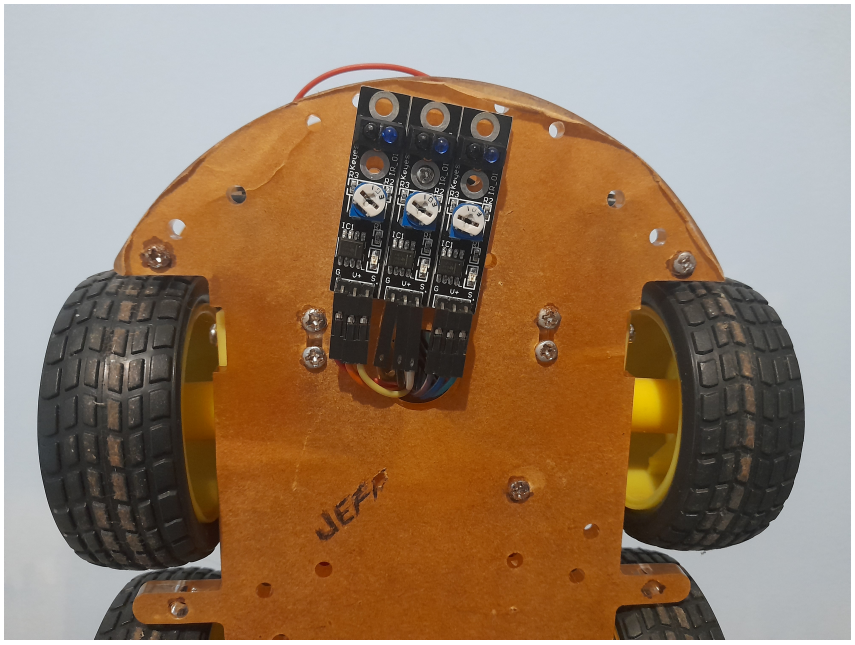


Figura 4 – Módulo sensor infravermelho.

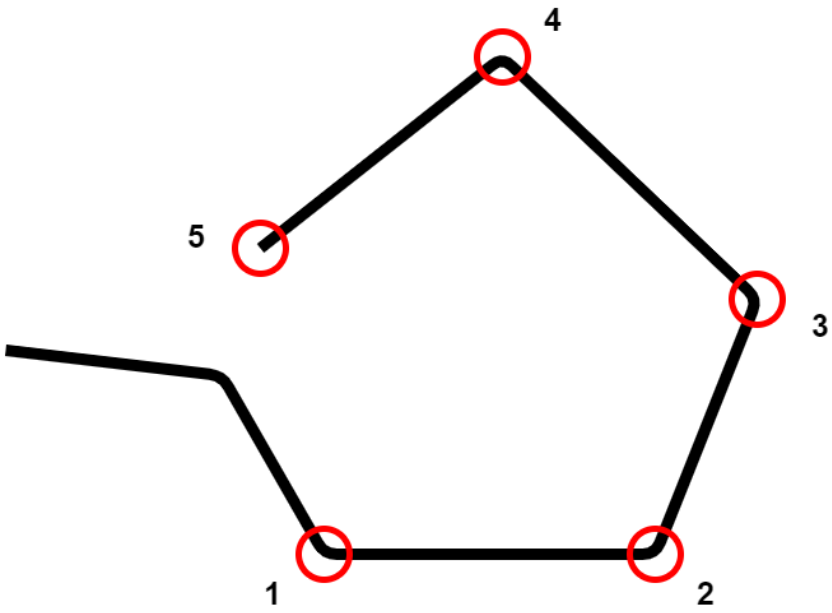


Figura 5 – Cenário 1 proposto para a realização dos experimentos.

5.2 Cenários

Para cada abordagem foram realizados 20 testes em sequência. Nos resultados das **Tabelas 3 e 4**, os *checkpoints* são identificados pela abreviação c + número (c1, c2, c3 ...), para todos os cenários.

Para o Método LF foi registrado apenas o último *checkpoint* efetivamente alcançado pelo robô antes de ele parar, seja por ter completado o percurso (c5 no Cenário 1 ou c4 no Cenário 2), seja por ter perdido a linha e não conseguir retornar ao trajeto. O tempo t corresponde ao intervalo, cronometrado manualmente em um segundo dispositivo, com-

preendido entre o instante em que o robô inicia movimento autônomo e o instante em que ele se imobiliza.

Para o método OD, em cada tentativa anota-se os *checkpoints* onde ocorreu colisão entre o robô e o objeto-alvo; quando não há colisão, nenhum *checkpoint* é marcado nessa coluna. Considera-se colisão o contato físico perceptível entre o chassi do robô e o objeto, verificado visualmente pelo observador. Esse evento pode resultar de erros na estimativa da distância ou do tamanho aparente do objeto, ocasionados por variações de iluminação, sombras, posição relativa ou ângulo da câmera, assim como pelas distorções introduzidas durante o deslocamento do robô. O cronômetro é acionado nas mesmas condições descritas para o LF e interrompido quando o robô para de se mover.

5.2.1 Cenário 1

Com o OD, o robô chegou ao final do percurso em todas as vezes, com exceção em uma tentativa conforme a (**Tabela 3**), em que o robô autônomo perdeu o objeto de vista e não conseguiu recuperar o rastreamento, e nesse caso o tempo não foi registrado. Não houve tentativas em que ocorreram mais que uma colisão. Pelo método LF, nota-se que, das 20 tentativas realizadas, em 15 delas o robô conseguiu percorrer todo o percurso e alcançar o último *checkpoint*.

Tabela 3 – Resultados dos experimentos (Cenário 1)

LF	Tentativas	Último *checkpoint* alcançado	Tempo(s)
	1	c3	10,19
	2	c5	10,21
	3	c5	10,05
	4	c2	16,45
	5	c5	10,84
	6	c5	11,23
	7	c5	10,98
	8	c5	10,32
	9	c5	18,41
	10	c4	18,81
	11	c5	19,73
	12	c5	8,88
	13	c5	9,08
	14	c5	9,47
	15	c5	11,31
	16	c5	9,53
	17	c5	9,54
	18	c1	12,28
	19	c2	30,87
	20	c5	11,3

OD	Tentativas	Colisão com o objeto	Tempo(s)
	1	c3	55,83
	2	c3	75,71
	3	-	106,62
	4	c3	-
	5	-	112,97
	6	c4	78,61
	7	-	68,51
	8	-	78,06
	9	c4	82,8
	10	-	52,4
	11	-	60,86
	12	-	57,22
	13	c3	68,74
	14	c2	61,58
	15	c3	78,64
	16	c5	66,22
	17	-	90,77
	18	-	100,81
	19	c1	71,92
	20	-	73,25

Comparando as duas formas de percorrer o trajeto, é possível perceber que o método OD, mesmo com tempos maiores e maior variação entre os tempos gastos, apre-

sentou um melhor desempenho em termos de sucesso na conclusão do percurso. Além de apresentar uma forma mais desafiadora e realista de percorrer o percurso.

5.2.2 Cenário 2

O cenário 2 foi elaborado com a hipótese de que o método LF falharia mais vezes, ao possuir curvas mais fechadas, a tese era que o método LF teria mais dificuldade de se manter na linha, enquanto que o método OD manteria a mesma performance. Esse novo percurso só possui 4 *checkpoints*, sendo o último o c4.

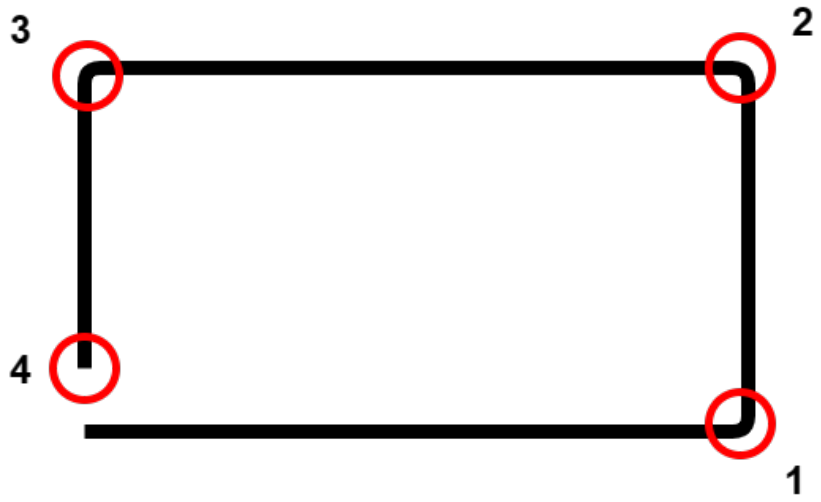


Figura 6 – Cenário 2.

Na tabela OD, houve tentativas em que houve mais de uma colisão, por isso foram registrados os *checkpoints* onde aconteceram as colisões, separados por vírgula.

O método LF concluiu o percurso com sucesso em 13 das 20 tentativas. O método OD concluiu o percurso com sucesso em todas as 20 tentativas, mesmo tendo ocorrido colisões com o objeto em 6 das tentativas o robô conseguiu se recuperar e alcançar o último *checkpoint*.

Comparando os resultados apresentados pelos cenários 1 e 2, é possível observar que o método LF teve um desempenho ligeiramente inferior no segundo cenário, com 13 tentativas bem-sucedidas contra 15 do primeiro cenário. Sendo assim, a hipótese de que o método LF teria piores resultados no segundo cenário se confirma. Embora seja necessária a realização de mais testes para obter uma conclusão mais robusta, os testes iniciais já seriam um indício da veracidade da hipótese.

5.3 Cenário com *smartphone* mais moderno

Até agora, os testes realizados utilizaram um *smartphone* com uma câmera de resolução de 2582 x 1936 *pixels* (Câmera 1). O objetivo desta seção é repetir os experimentos

Tabela 4 – Resultados dos experimentos (Cenário 2)

LF	Tentativas	Último *checkpoint* alcançado	Tempo(s)
	1	c1	2,83
	2	c2	4,8
	3	c4	9,08
	4	c4	11,5
	5	c2	5,77
	6	c4	8,5
	7	c4	18,42
	8	c4	7,54
	9	c4	8,81
	10	c4	10,66
	11	c4	8,18
	12	c1	2,5
	13	c2	4,32
	14	c4	21,58
	15	c4	20,22
	16	c4	8,06
	17	c4	14,45
	18	c1	2,47
	19	c4	12,31
	20	c1	2,17

OD	Tentativas	Colisão com o objeto	Tempo(s)
	1	-	101,72
	2	c1,c2	88,67
	3	-	104,84
	4	-	152,80
	5	-	79,19
	6	-	70,25
	7	-	83,03
	8	-	83,64
	9	-	109,82
	10	-	81,37
	11	c1	87,86
	12	-	86,10
	13	-	102,47
	14	-	77,86
	15	-	88,26
	16	c4	110,38
	17	c3	94,64
	18	c2	122,89
	19	-	150,30
	20	c1	186,99

com um novo *smartphone* que tem maior capacidade de processamento (**Tabela 5**) e uma câmera com resolução de 8000 x 6000 *pixels* (Câmera 2). Para avaliar de forma objetiva as melhorias que uma câmera de melhor resolução pode trazer, foi conduzido um teste para comparar as medidas obtidas pela câmera com as medidas reais, uma vez que o experimento principal considera a distância entre a câmera e o objeto. Esse teste foi realizado para cada câmera.

Para assegurar comparabilidade, manteve-se todo o aparato experimental: o *smartphone* permaneceu acoplado ao chassi do robô; usou-se o mesmo objeto dos ensaios anteriores, posicionado no mesmo ambiente, sob a mesma iluminação; adotou-se o mesmo aplicativo e os mesmos parâmetros de calibração. A única alteração no código foi redirecionar a saída, após o cálculo da média das distâncias, o valor passou a ser exibido na tela em vez de ser enviado como comando de navegação. O procedimento consiste em posicionar o conjunto robô + câmera em distâncias de 5 cm a 210 cm do objeto, em incrementos de 5 cm. Em cada posição o sistema permaneceu ativo por um tempo arbitrário que varia entre 30 segundos e 3 minutos, intervalo suficiente para que fosse observado e registrado a menor e maior distância estimada pelo aplicativo. Após a execução de todos os testes, foi calculado o erro para cada estimativa como $d_{reaal} - d_{estimada}$. A **Tabela 6** consolida esses resultados para a Câmera 1, cada linha indica a distância real, os valores mínimo e máximo estimados e o erro correspondente. O mesmo protocolo foi, então, repetido para a Câmera 2, permitindo a comparação direta do desempenho entre as duas configurações de hardware.

Especificação	**Smartphone 1 (Samsung Galaxy J1 Mini)**	**Smartphone 2 (Samsung Galaxy M12)**
Modelo	Samsung Galaxy J1 Mini	Samsung Galaxy M12
Sistema Operacional	Android 5.1.1 (Lollipop)	Android 11, One UI Core 3.1
CPU	Quad-core 1.2 GHz Cortex-A7	Octa-core (4×2.0 GHz Cortex-A55 & 4×2.0 GHz Cortex-A55)
GPU	Mali-400MP2	Mali-G52
Memória (RAM)	768 MB	4 GB
Câmera Principal	5 MP (2582 × 1936)	48 MP (8000 × 6000)

Fontes: (MILINOV, 2015), (TUDOCELULAR.COM, 2016), (MILINOV, 2021), (TUDOCELULAR.COM, 2021).

Tabela 5 – Comparação das especificações técnicas dos smartphones utilizados

Com base nos resultados apresentados, é possível observar que a câmera apresentou uma boa estimativa da distância em alguns casos, enquanto em outros apresentou uma grande variação em relação à distância real. Por exemplo, para a distância de 45 centímetros, a câmera estimou uma distância entre 42 e 46 centímetros. Já para a distância de 145 centímetros, a câmera estimou uma distância entre 178 e 272 centímetros, com erros na estimativa de -33 e -127 centímetros, respectivamente. Em geral, pode-se afirmar que os erros das estimativas tendem a aumentar à medida que a distância aumenta.

Além disso, o desvio padrão apresenta variações de 92,83 para a menor distância e 111,22 para a maior distância. Porém, considerando apenas os primeiros 20 resultados, ou seja, para distâncias menores que 1 metro, é possível verificar que o desvio padrão dos erros é menor, ficando em 7,42 para a menor distância e 7,12 para a maior distância. Isso indica que, apesar de os erros da câmera serem bastante variáveis, para distâncias menores que 100 centímetros a variação mantém uma certa estabilidade.

Tabela 6 – Testes Câmera 1

Distância	**Menor Estimado**	**Maior Estimado**	**Erro Menor Estimativa**	**Erro Maior Estimativa**
5	9	10	-4	-5
10	18	30	-8	-20
15	18	30	-3	-15
20	30	38	-10	-18
25	32	36	-7	-11
30	34	43	-4	-13
35	37	45	-2	-10
40	36	40	4	0
45	42	46	3	-1
50	50	57	0	-7
55	48	57	7	-2
60	50	55	10	5
65	52	61	13	4
70	67	76	3	-6
75	64	70	11	5
80	70	82	10	-2
85	83	94	2	-9
90	78	90	12	0
95	82	97	13	-2
100	88	103	12	-3
105	99	125	6	-20
110	123	150	-13	-40
115	120	146	-5	-31
120	136	166	-16	-46
125	119	151	6	-26
130	150	178	-20	-48
135	152	195	-17	-60
140	162	228	-22	-88
145	178	272	-33	-127
150	193	304	-43	-154
155	208	296	-53	-141
160	224	312	-64	-152
165	266	365	-101	-200
170	308	426	-138	-256
175	319	447	-144	-272
180	387	466	-207	-286
185	432	477	-247	-292
190	424	488	-234	-298
195	445	488	-250	-293
200	452	488	-252	-288
205	472	488	-267	-283
210	477	477	-267	-267

Desvio Padrão dos erros (menor distância): 92,83
Desvio Padrão dos erros (maior distância): 111,22

Desvio Padrão dos erros (menor distância - primeiros 20): 7,42
Desvio Padrão dos erros (maior distância - primeiros 20): 7,12

Tabela 7 – Testes Câmera 2

Distância	Menor Estimado	Maior Estimado	Erro Menor Estimativa	Erro Maior Estimativa
5	4	6	1	-1
10	10	13	0	-3
15	14	17	1	-2
20	14	16	6	4
25	13	14	12	11
30	26	28	4	2
35	28	33	7	2
40	27	44	13	-4
45	29	53	16	-8
50	41	58	9	-8
55	39	56	16	-1
60	35	57	25	3
65	43	57	22	8
70	50	82	20	-12
75	61	78	14	-3
80	55	66	25	14
85	68	81	17	4
90	75	92	15	-2
95	63	80	32	15
100	105	119	-5	-19
105	176	233	-71	-128
110	139	165	-29	-55
115	217	319	-102	-204
120	154	216	-34	-96
125	155	195	-30	-70
130	390	488	-260	-358
135	112	149	23	-14
140	146	209	-6	-69
145	171	215	-26	-70
150	301	450	-151	-300
155	164	247	-9	-92
160	183	236	-23	-76
165	183	195	-18	-30
170	172	228	-2	-58
175	146	202	29	-27
180	323	377	-143	-197
185	207	217	-22	-32
190	187	200	3	-10
195	223	328	-28	-133
200	383	488	-183	-288
205	181	259	24	-54
210	488	488	-278	-278

Desvio Padrão (menor distância) 72,65
Desvio Padrão (maior distância) 94,95

A **Tabela 7** apresenta uma ligeira melhora nos resultados. Os dados obtidos pela Câmera 1 mostram uma maior variabilidade nas estimativas de distância em comparação com a Câmera 2, o que é denotado pelo Desvio Padrão. Isso indica que a Câmera 1 pode não ser tão precisa quanto a Câmera 2. Também é de se observar que, considerando apenas os primeiros 20 resultados, os desvios-padrão dos erros da menor e maior distância da Câmera 1 são menores que os da Câmera 2.

É importante salientar que para este trabalho, em um primeiro momento é mais importante a detecção do objeto do que a precisão em estimar a distância, sendo que não é possível aferir alguma distância se não houver detecção do objeto. Isso é uma característica que não fica evidente nos resultados das **Tabelas 6** e **7**.

Destrinchando melhor os dados obtidos pela Câmera 1, para as distâncias acima de 150 centímetros, em alguns momentos a câmera não conseguiu detectar o objeto a fim de aferir sua distância, e na distância de 210 centímetros a câmera só conseguiu detectar o objeto uma única vez, por isso a menor e maior distância estimada possuem o mesmo valor. Essa complicação não aconteceu com a Câmera 2, onde a detecção ocorreu em todas as distâncias sem nenhuma falha, além de conseguir estimar mais distâncias em menos tempo.

Em resumo, a Câmera 2 apresenta resultados mais estáveis, como apontam os desvios padrão, é capaz de detectar o objeto em distâncias maiores e é mais rápida para estimar as distâncias. Uma observação a ser feita é que, embora a Câmera 1 tenha demonstrado uma variabilidade menor nas estimativas em comparação à Câmera 2 para distâncias menores que 1 metro, esse resultado pode ser alcançado pela Câmera 2 com uma calibração mais precisa.

Com os novos resultados obtidos (**Tabela 8**) foi elaborada uma tabela comparativa (**Tabela 9**), onde são apresentados de forma resumida os resultados do método OD nos dois cenários para cada Câmera, isto é, os resultados da Primeira Câmera nos Cenários 1 e 2 (O_C1 e O_C2); e os resultados da Segunda Câmera nos Cenários 1 e 2 (N_C1 e N_C2). Ao analisar a **Tabela 9**, é possível observar que, para o Cenário 1, houve melhoria no tempo médio (de 75,87 para 71,44 segundos), enquanto o número de colisões e o desvio padrão aumentaram, indicando uma possível inconsistência apesar da melhoria do tempo. Já para o Cenário 2, o que se nota é uma expressiva e sólida melhoria em todos os aspectos, com a média do tempo caindo quase pela metade.

Tabela 8 – Resultados dos experimentos (Cenários 1 e 2)

C1	Tentativas	Colisão com o objeto	Tempo(s)
	1	-	54,62
	2	-	52,79
	3	c5	62,02
	4	c1,c5	49,03
	5	-	53,10
	6	c5	66,70
	7	c5	55,60
	8	c5	56,24
	9	c5	85,56
	10	c4,c5	61,37
	11	c5	59,34
	12	-	60,71
	13	-	65,65
	14	-	66,68
	15	-	132,42
	16	-	93,78
	17	-	66,51
	18	c5	108,50
	19	c1	102,45
	20	c1	75,63

C2	Tentativas	Colisão com o objeto	Tempo(s)
	1	-	46,18
	2	-	60,25
	3	c4	53,67
	4	-	48,22
	5	-	51,37
	6	-	47,60
	7	-	74,56
	8	-	39,67
	9	-	40,21
	10	-	72,89
	11	-	52,84
	12	-	53,16
	13	-	43,10
	14	-	45,06
	15	-	60,54
	16	-	49,33
	17	-	57,23
	18	-	67,08
	19	-	80,56
	20	c4	66,43

Tabela 9 – Comparação de Resultados

Tabela	**Número de Colisões**	**Média do Tempo (s)**	**Desvio Padrão do Tempo (s)**
O_C1	10	75,87	16,51
O_C2	7	103,15	28,97
N_C1	13	71,44	21,53
N_C2	2	55,50	11,44

5.4 Explorando características do *smartphone* atual

Os dados das **Tabelas 6** e **7** foram representados em gráficos (**Figuras 7** e **8**) para facilitar uma análise visual. Além disso, foi elaborada um gráfico de dispersão (**Figura 9**), comparando os dois resultados. Observa-se que, na Câmera 1, os erros aumentam de forma quase linear, conforme o esperado. Em contrapartida, os resultados obtidos com a Câmera 2 exibem picos, com valores extremamente altos seguidos por quedas abruptas. Esse comportamento pode ser atribuído à presença de recursos avançados, como o foco automático, que não estão presentes na Câmera 1, ocasionando uma variabilidade atípica nos resultados.

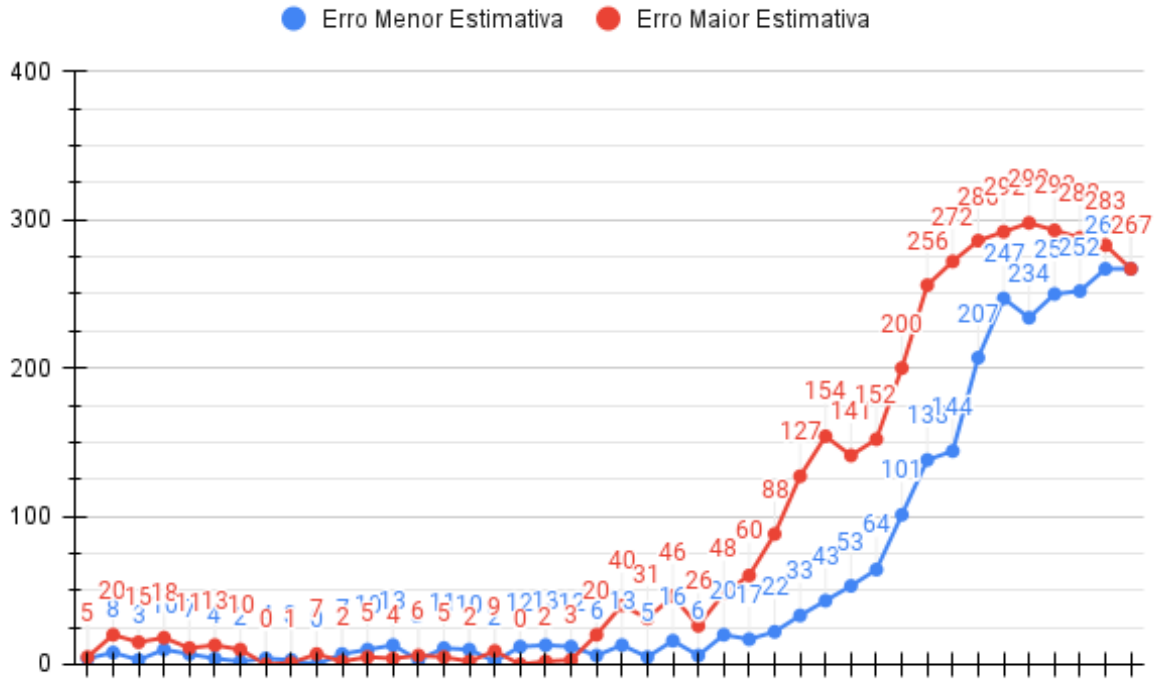


Figura 7 – Erros Câmera 1.

Para explorar essa característica específica da Câmera 2, foram implementadas alterações no código com o intuito de reduzir os *spikes* observados. Foram testadas quatro abordagens distintas: duas delas consistem em aumentar o número de imagens utilizadas para o cálculo da média das distâncias, de forma que os valores extremos sejam "diluídos"; as outras duas abordagens visam eliminar *outliers* antes do cálculo da média final, além de aumentar a quantidade de imagens analisadas. Vale destacar que esse processo só é viável graças à capacidade computacional do novo *smartphone*, que permite a análise de um maior número de imagens com maior rapidez.

Na primeira abordagem (**Figura 10**), onde a média das distâncias foi calculada a partir de 30 imagens em vez de 15, observou-se uma redução significativa no número de picos (4 em comparação com 7). Além disso, os picos passaram a ocorrer a partir de 130

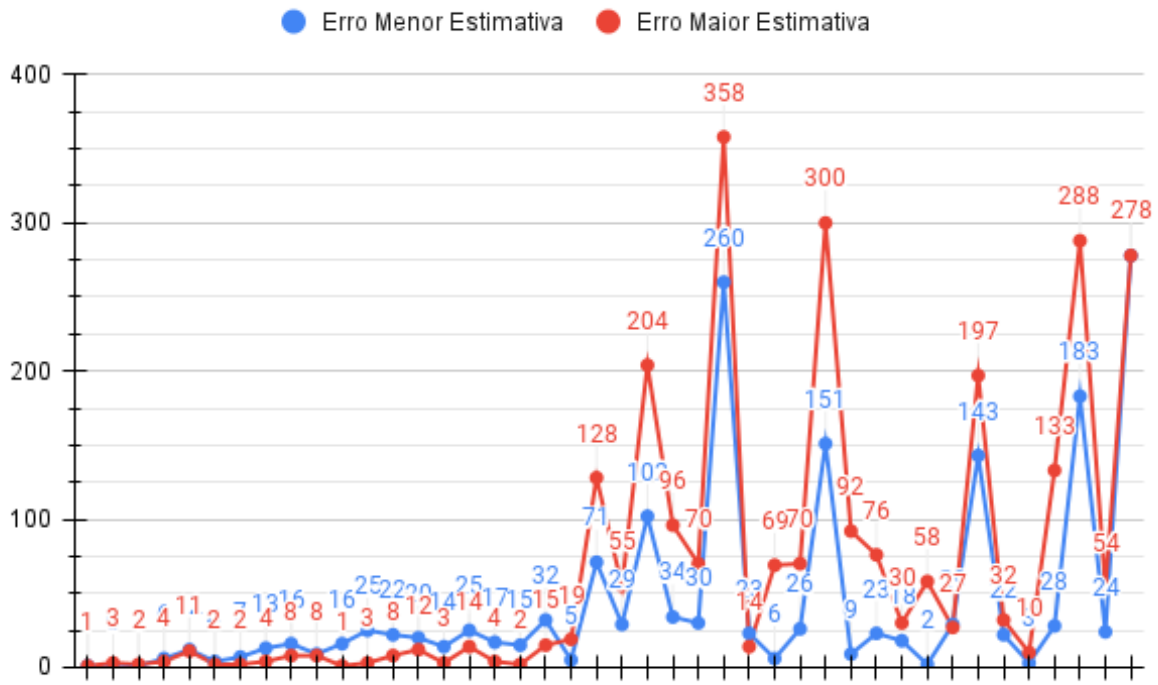
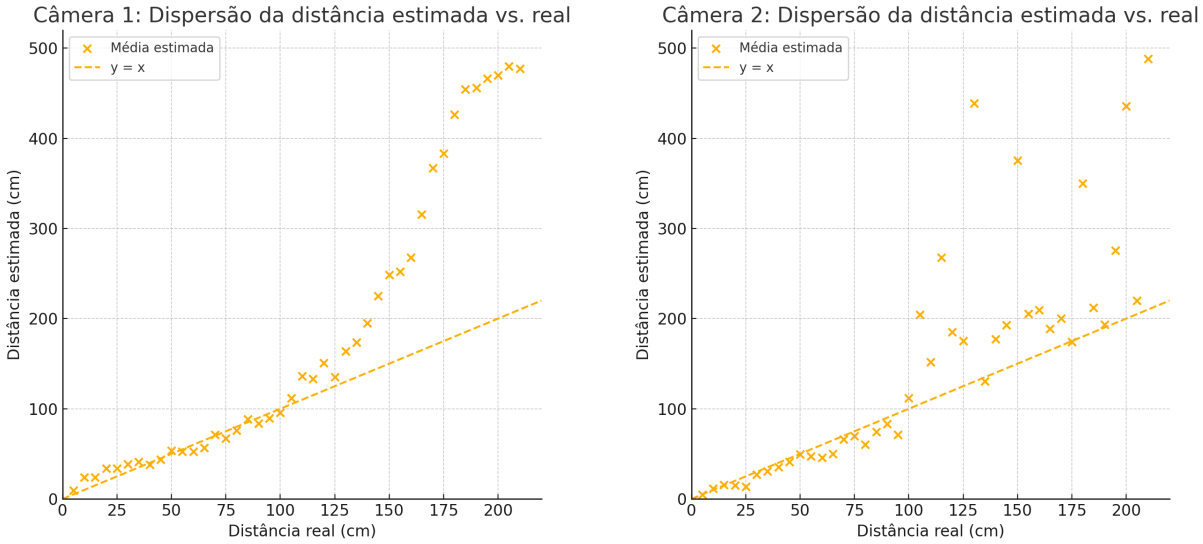


Figura 8 – Erros Câmera 2.



(a) Câmera 1

(b) Câmera 2

Figura 9 – Dispersão das distâncias estimadas em função da distância real para as duas câmeras. A linha tracejada indica concordância perfeita ($y = x$).

centímetros, enquanto anteriormente surgiam a partir dos 100 centímetros. A redução dos *spikes* também resultou na diminuição dos desvios padrão, de 72,65 para 49,96 e de 94,95 para 78,83, para os erros das menores e maiores distâncias, respectivamente.

Os resultados apresentados na **Figura 11** foram obtidos com a média de 60 distâncias. Observa-se que os picos ainda se formam, porém de maneira menos acentuada,

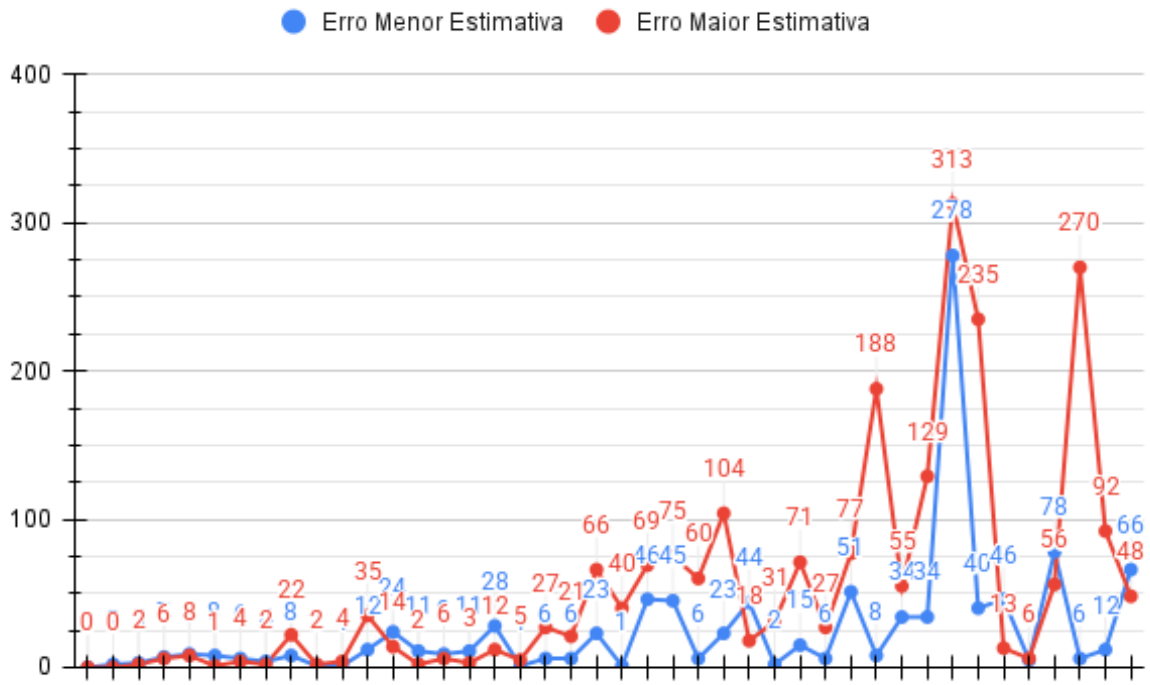


Figura 10 – Abordagem: média das distâncias de 30 imagens.

indicando que, à medida que aumenta o número de distâncias consideradas, as estimativas se aproximam mais do valor real. De fato, ao comparar os testes realizados com 30 e 60 distâncias, houve uma redução expressiva nos desvios padrão dos erros. Enquanto o experimento com 30 distâncias apresentou desvios padrão de 49,96 para os erros da menor distância estimada e 78,83 para os da maior distância, com 60 distâncias esses valores foram reduzidos para 26,99 e 42,09, respectivamente, evidenciando uma maior consistência dos dados.

No entanto, o aumento no número de amostras resulta em um maior tempo de processamento, o intervalo entre os resultados chegou a valores entre 2 e 3 segundos, o que poderia comprometer a agilidade necessária para aplicações em tempo real. Dado que o robô realiza sua ação a cada 1 segundo, seria adequado manter esse intervalo em aproximadamente 1 segundo. Assim, para obter uma análise mais ágil utilizando 60 distâncias, mantendo a precisão das medições sem sacrificar a responsividade, seria necessário realizar otimizações no código ou implementar melhorias no *hardware*.

O próximo experimento (**Figura 12**) consiste em incluir um mecanismo de filtragem estatística para reduzir a influência de valores discrepantes na estimativa da distância. Considerando 30 distâncias estimadas, calcula-se a média e o desvio padrão, com isso é definido um intervalo (média ± desvio padrão). Apenas as medições que estejam dentro desse intervalo são utilizadas para recalcular uma média filtrada, que se espera ser mais robusta e representativa do valor real, eliminando *spikes* e valores anômalos.

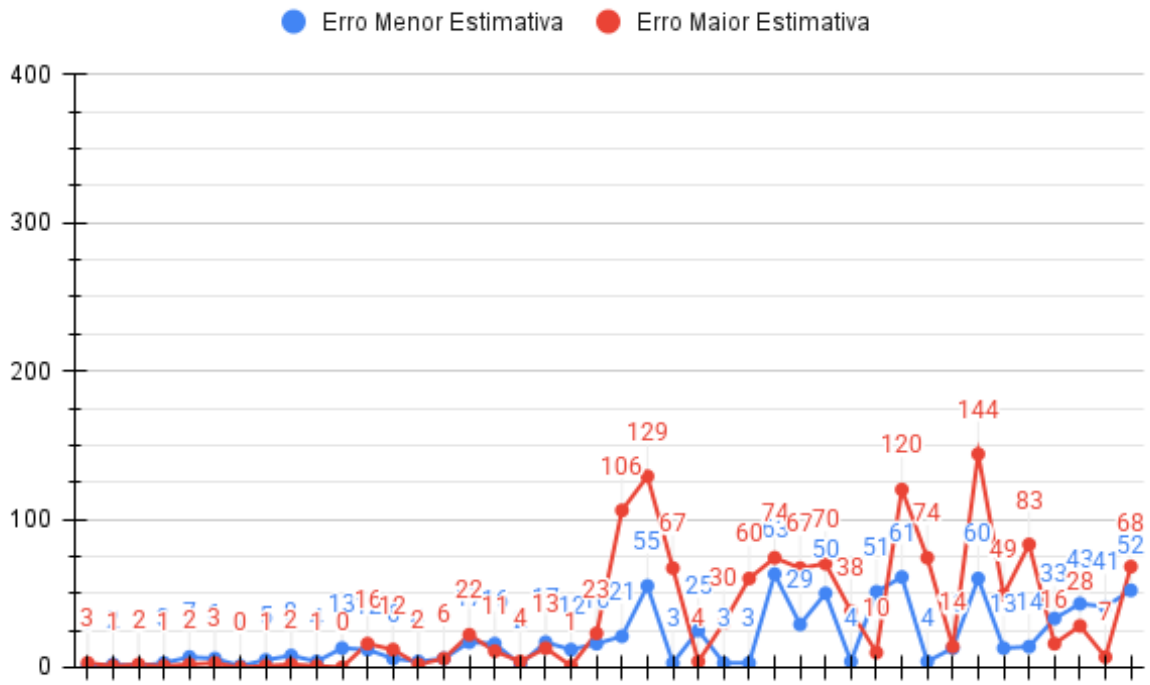


Figura 11 – Abordagem: média das distâncias de 60 imagens.

Ao analisar os resultados do experimento, constata-se uma melhora moderada em comparação aos testes que utilizaram somente a média de 30 distâncias. Verificou-se uma redução dos desvios padrão: para os erros da menor distância estimada, o desvio padrão diminuiu de 49,96 para 43,75, e para os erros da maior distância, de 78,83 para 65,35. A **Figura 12** ilustra a atenuação de alguns picos. Entretanto, é importante ressaltar que, devido ao método de eliminação dos *outliers*, pode ocorrer que nenhuma das 30 distâncias obtidas esteja dentro do intervalo estipulado, o que merece atenção.

Com isso chega-se na próxima abordagem que consiste em fazer a eliminação de *outliers* pelo método de Intervalo Entre Quartis (IQR). Ele é baseado na divisão dos dados em quartis, que representam pontos que separam os valores em quatro partes iguais. O IQR é a diferença entre o terceiro e o primeiro quartil ($Q3 - Q1$), esse intervalo mede a dispersão central dos dados, ignorando os valores extremos. Para identificar e eliminar *outliers*, define-se um limite inferior e um limite superior, da seguinte forma:

$$LimiteInferior = Q1 - 1.5 * IQR$$

$$LimiteSuperior = Q3 + 1.5 * IQR$$

Assim, os valores de distâncias que estiverem fora desse intervalo são considerados *outliers*, pois diferem significativamente do padrão geral dos dados e não são considerados no cálculo da média final.

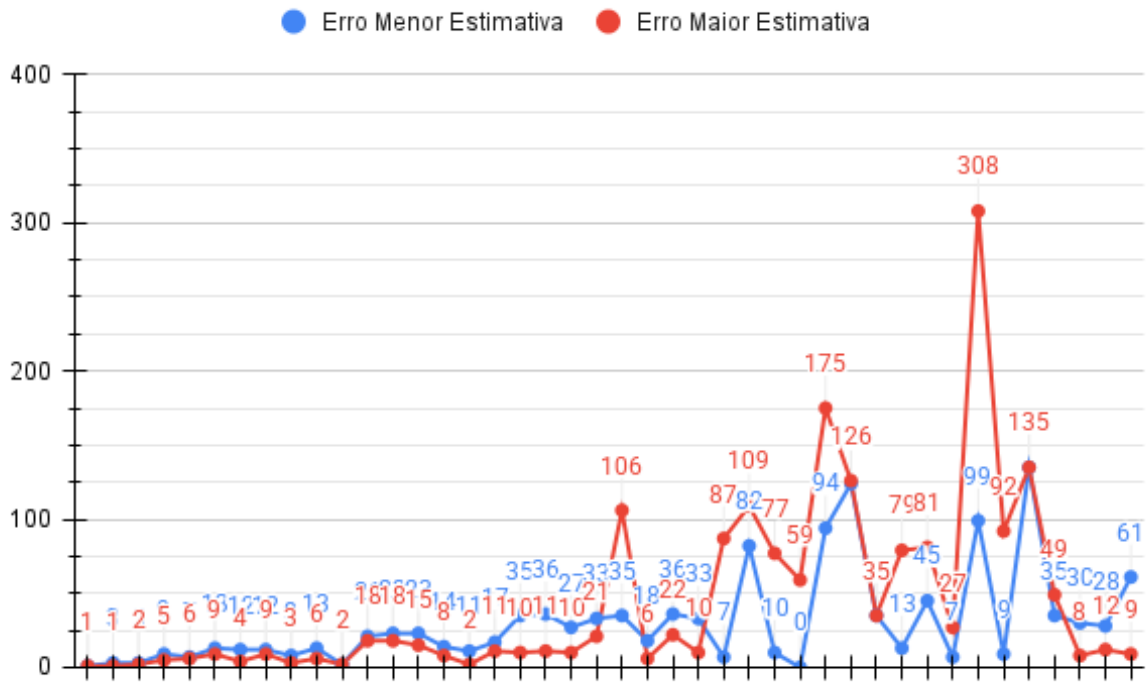


Figura 12 – Abordagem: eliminação de *outliers*, 30 distâncias.

Os resultados apresentados na **Figura 13** demonstram que os experimentos utilizando o método IQR se aproximam do desempenho dos testes realizados com 60 médias de distâncias (**Figura 11**), que até então obteve os melhores resultados. Especificamente, os desvios padrão dos erros foram de 33,49 para a menor distância e 55,27 para a maior distância, em comparação com 26,99 e 42,09, respectivamente, para o método de 60 médias. Assim, o método IQR quase alcança a performance do método que utiliza 60 médias.

Além disso, o tempo de processamento entre cada média final permaneceu em torno de 1 segundo, o que é crucial para a navegação em tempo real do robô. É de se esperar que a combinação dessa abordagem estatística com 60 médias de distâncias possa resultar em uma melhoria ainda mais significativa dos resultados, embora a questão do tempo de processamento exija otimizações e/ou *upgrades* de *hardware*. A **Figura 14** consolida, lado a lado, as quatro abordagens descritas nas **Figuras 10 - 13**.

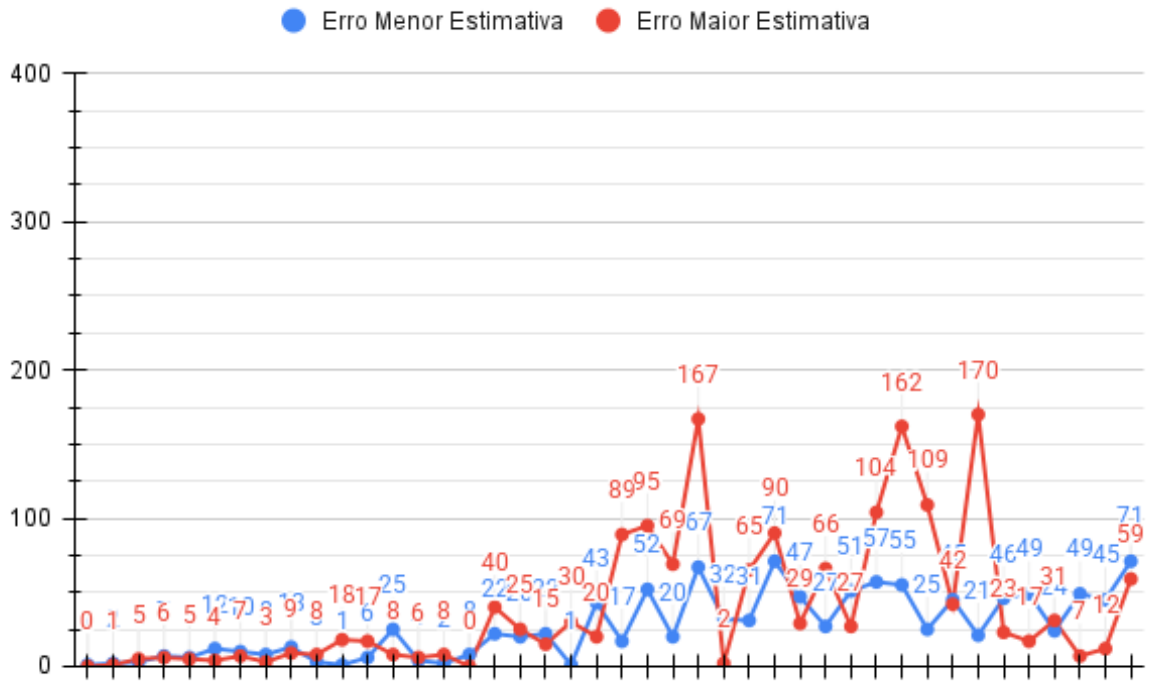


Figura 13 – Abordagem: eliminação de *outliers* pelo método IQR , 30 distâncias.

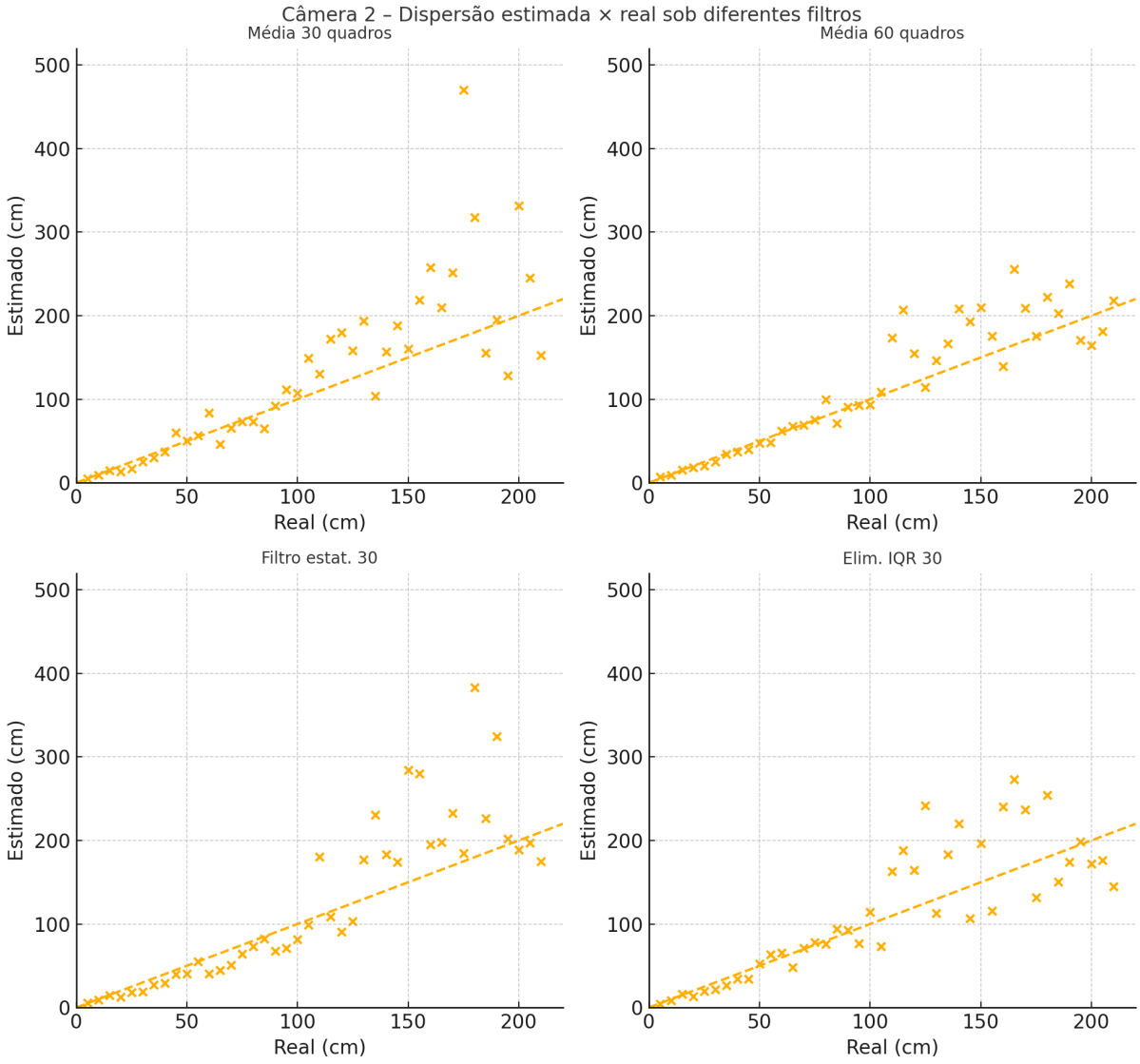


Figura 14 – Câmera 2: dispersão entre distância real e estimada sob quatro abordagens de filtragem. A linha tracejada indica concordância perfeita ($y = x$).

6 Conclusão

Este trabalho introduziu e validou uma integração entre *smartphone* e kit robótico de baixo custo que, com auxílio do OpenCV embarcado, executa detecção de objetos e navegação autônoma sem qualquer sensor dedicado. Nos dois cenários propostos, o sistema demonstrou confiabilidade superior ao *line-follower* tradicional. O método seguidor de linha (LF) é notoriamente veloz, e superou o OD em relação ao tempo, porém é um método sem flexibilidade, que restringe a movimentação do robô autônomo a seguir a linha, e que exige uma manipulação prévia do ambiente na demarcação das linhas bem definidas. Somado a isso, falhas no LF são catastróficas; se o robô sai da linha, é muito difícil retomar o percurso, e o LF não foi capaz de chegar ao último *checkpoint* em várias tentativas realizadas nos experimentos, contra 1 de 20 do OD proposto no cenário 1, e nenhuma no cenário 2.

Os resultados indicam que o método OD é mais confiável do que o método LF para concluir com sucesso um percurso com *checkpoints*, mesmo que o método LF seja mais rápido em algumas tentativas bem-sucedidas. O método LF é limitado pelo fato de depender da linha desenhada no chão e ter menos capacidade de adaptação a imprevistos.

Os novos experimentos conduzidos com uma câmera de maior resolução indicaram que o método OD poderia ter melhores resultados. Esse aumento da resolução permitiu que o algoritmo *Blob Detection* identificasse o objeto em todas as distâncias avaliadas. No entanto, foram observados falsos positivos durante os testes, que seria explicado pelo aumento da resolução, apesar de não ter comprometido o experimento. O simples incremento na capacidade computacional abre perspectivas promissoras para aprimoramentos no algoritmo, como demonstrado nos experimentos para diminuir a oscilação no gráfico dos erros de menor e maior distância estimada. Os quatro filtros estatísticos propostos oferecem opções graduais de compromisso entre latência e precisão, abrindo caminhos para futuras otimizações.

Assim, o OD é mais versátil, mais tolerante a falhas e bastante acessível conforme os experimentos realizados. Um resultado importante deste trabalho foi propor uma arquitetura de baixo custo de visão computacional com um kit de robótica baseado em Arduino e um celular Android, e demonstrar ser possível realizar com ela tarefas complexas de processamento de imagens e navegação robótica. Assim, o custo do kit e a acessibilidade do celular Android usado permitem que ele seja utilizado em salas de aula e aplicações de múltiplos robôs, que são uma possibilidade interessante para trabalhos futuros. Já que aplicações multi-robôs apresentam alto custo para serem implementadas. O desempenho das câmeras de diferentes celulares Android na aplicação de detecção e rastreamento de

objetos também é uma análise futura com potencial interessante.

Referências

ALYMANI, M.; KARAR, M.; SHEHATA, H. A practical study of intelligent image-based mobile robot for tracking colored objects. *Computers, Materials Continua*, v. 80, 08 2024. Citado 2 vezes nas páginas 12 e 14.

Android Developers. *Platform Architecture*. 2024. Acessado em 22/05/2025. Disponível em: <https://source.android.com/docs/core/architecture>. Citado 2 vezes nas páginas 3 e 9.

ARDUINO. 2021. <https://www.arduino.cc/en/Guide/Introduction>. Acessado em 14/06/2021. Citado na página 10.

Backlinko Team. *Smartphone Usage Statistics (2025)*. 2025. Acessado em 15/05/2025. Disponível em: <https://backlinko.com/smartphone-usage-statistics>. Citado 2 vezes nas páginas 4 e 7.

BASLER AG. *Basler ace Camera*. 2025. <https://hr.mouser.com/c/optoelectronics/cameras-accessories/cameras-camera-modules/?m=Basler&series=Ace&>. Preço consultado em 17 mai. 2025. Citado na página 6.

BRADSKI, G.; KAEHLER, A. *Learning OpenCV: Computer vision with the OpenCV library*. [S.l.]: "O'Reilly Media, Inc.", 2008. Citado na página 10.

DALAL, N.; TRIGGS, B. Histograms of oriented gradients for human detection. *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, v. 1, p. 886–893 vol. 1, 2005. Disponível em: <https://api.semanticscholar.org/CorpusID:206590483>. Citado na página 11.

DESOUZA, G. N.; KAK, A. C. Vision for mobile robot navigation: A survey. *IEEE transactions on pattern analysis and machine intelligence*, IEEE, v. 24, n. 2, p. 237–267, 2002. Citado na página 10.

FRANçA, D. *Vídeo 1 – Navegação Line Follower*. 2025. Google Drive. Duração: 13 seg. Disponível em: <https://drive.google.com/file/d/1ctlxx3AuJSwTpluPSVNLGhfRM80RtkqA/view>. Citado na página 19.

FRANçA, D. *Vídeo 2 – Navegação Object Detection*. 2025. Google Drive. Duração: 1 min. Disponível em: <https://drive.google.com/file/d/1gt5AhdgVkYK-7t2IcrkB6-JxTkrOm1nG/view>. Citado na página 19.

FRANçA, D. V. *Color Object Detection*. 2025. <https://doi.org/10.5281/zenodo.15337983>. Disponível em: <https://github.com/Danilo11Franca/ColorObjectDetection>. Citado na página 15.

FRANçA, D. V. *Robot Navigation*. 2025. <https://doi.org/10.5281/zenodo.15338028>. Disponível em: <https://github.com/Danilo11Franca/robot-navigation>. Citado na página 15.

FUSCO, R. J.; ASSIS, W. O. Desenvolvimento de um sistema de visão para robô. 2015. Citado 2 vezes nas páginas 12 e 14.

GONZALEZ, R.; WOODS, R. *Digital Image Processing*. [S.l.]: Prentice Hall, 2002. Citado na página 9.

HAN, N.; RYU, S. J.; NAM, Y. Real-time moving object tracking on smartphone using cradle head servo motor. *Sensors*, v. 24, n. 4, 2024. ISSN 1424-8220. Disponível em: <https://www.mdpi.com/1424-8220/24/4/1265>. Citado 2 vezes nas páginas 13 e 14.

HARTLEY, R. I.; ZISSERMAN, A. *Multiple View Geometry in Computer Vision*. Second. [S.l.]: Cambridge University Press, ISBN: 0521540518, 2004. Citado na página 11.

INTEL CORPORATION. *Intel RealSense Depth Camera*. 2025. <https://store.intelrealsense.com/?srsltid=AfmBOoqCSXqeJ0DjeVSCLVzIJJ0NBbQkY9KewtLgpmwAjJfRpaWlwEVH>. Preço consultado em 17 mai. 2025. Citado na página 6.

KING, B. *Is Android Really Open Source? And Does It Even Matter?* 2016. Acessado em 14/06/2021. Disponível em: <https://www.makeuseof.com/tag/android-really-open-source-matter/>. Citado na página 9.

LIU, L. et al. *Deep Learning for Generic Object Detection: A Survey*. 2019. Disponível em: <https://arxiv.org/abs/1809.02165>. Citado na página 11.

MAKERFOCUS. *ESP32-CAM WiFi/Bluetooth Module with OV2640 Camera*. 2025. <https://www.makerfocus.com/products/esp32-camera-wifi-bluetooth-module?srsltid=AfmBOoqRVn4nDViNfqWmKgppaR_JMG3tBcHzZuB_PnznJls0s4AtJghA>. Preço consultado em 17 mai. 2025. Citado na página 6.

MCROBERTS, M. *Beginning Arduino*. [S.l.]: Apress, 2011. Citado 2 vezes nas páginas 3 e 10.

MCROBERTS, M. *Arduino básico*. [S.l.]: Novatec Editora, 2018. Citado na página 10.

MENDOZA, D. Computer vision applied to autonomous robots using raspberry pi. 2014. Citado 2 vezes nas páginas 12 e 14.

MILINOV, O. *Samsung Galaxy J1 Nxt - Full phone specifications*. 2015. Acessado em 26/03/2025. Disponível em: <https://www.gsmarena.com/samsung_galaxy_j1_nxt-7768.php>. Citado na página 24.

MILINOV, O. *Samsung Galaxy M12 - Full phone specifications*. 2021. Acessado em 23/04/2025. Disponível em: <https://www.gsmarena.com/samsung_galaxy_m12-10860.php>. Citado na página 24.

MORETTO; MELLO. Protótipo de um robô seguidor de piso tátil utilizando visão. 2020. Citado 2 vezes nas páginas 12 e 14.

MORICH, K. *Serial Bluetooth Terminal*. 2021. Disponível em: <https://play.google.com/store/apps/details?id=de.kai_morich.serial_bluetooth_terminal>. Citado na página 16.

NANDHINI, R.; AL-KHAN, M.; KUMAR, S. Arduino-based smart-phone controlled robot car. *International Journal of Science and Advanced Technology*, v. 5, n. 4, p. 34–38, 2025. Disponível em: <<https://www.ijsat.org/research-paper.php?id=3814>>. Citado 2 vezes nas páginas 13 e 14.

OROS; JEFFREY. Smartphone based robotics: Powerful, flexible and inexpensive robots for hobbyists, educators, students and researchers. *CECS Tech Report 13-16*, 2013. Citado 2 vezes nas páginas 12 e 14.

PEREIRA, L. C. O.; SILVA, M. L. da. *Android para desenvolvedores*. [S.l.]: Brasport, 2009. Citado na página 9.

RASPBERRY PI FOUNDATION. *Raspberry Pi High Quality Camera (Sony IMX477)*. 2025. <https://www.raspberrypi.com/products/raspberry-pi-high-quality-camera/>. Preço consultado em 17 mai. 2025. Citado na página 6.

REDMON, J. et al. You only look once: Unified, real-time object detection. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*. IEEE Computer Society, 2016. p. 779–788. Disponível em: <https://doi.org/10.1109/CVPR.2016.91>. Citado na página 11.

ROMERO, R. et al. Robótica móvel, editora ltc. 2014. Citado na página 15.

SEMICONDUCTORS, V. *Application of Optical Reflex Sensors: TCRT1000, TCRT5000, CNY70*. [S.l.], 2023. Tabela 1 traz refleção relativa: papel branco 100%, tinta preta 4–6%. Disponível em: <https://www.vishay.com/docs/80107/80107.pdf>. Citado na página 19.

SLAMTEC. *SLAMTEC RPLIDAR A1M8 360° Laser Scanner*. 2025. <https://www.robotshop.com/products/rplidar-a1m8-360-degree-laser-scanner-development-kit?srsltid=AfmBOoqs6g_DsH-FLEeAZYOrfGhglRbNPDyWbxAuLM13Bj95rEfqg5vm>. Preço consultado em 17 mai. 2025. Citado na página 6.

STEIN, G. et al. A novice-friendly and accessible networked educational robotics simulation platform. *Education Sciences*, v. 15, n. 2, 2025. ISSN 2227-7102. Disponível em: <https://www.mdpi.com/2227-7102/15/2/198>. Citado 3 vezes nas páginas 6, 13 e 14.

THRUN, S.; BURGARD, W.; FOX, D. *Probabilistic Robotics*. MIT Press, 2005. (Intelligent Robotics and Autonomous Agents series). ISBN 9780262201629. Disponível em: <https://books.google.com.br/books?id=2Zn6AQAAQBAJ>. Citado na página 10.

TUDOCELULAR.COM. *Samsung Galaxy J1 Mini - Ficha Técnica - TudoCelular.com*. 2016. Acessado em 26/03/2025. Disponível em: <https://www.tudocelular.com/Samsung/fichas-tecnicas/n3162/Samsung-Galaxy-J1-Mini.html>. Citado na página 24.

TUDOCELULAR.COM. *Samsung Galaxy J1 Mini - Ficha Técnica - TudoCelular.com*. 2021. Acessado em 23/04/2025. Disponível em: <https://www.tudocelular.com/Samsung/fichas-tecnicas/n6838/Samsung-Galaxy-M12.html>. Citado na página 24.

VIOLA, P.; JONES, M. Rapid object detection using a boosted cascade of simple features. *Comput. Vis. Pattern Recog*, v. 1, 01 2001. Citado na página 11.