

Universidade Federal do Piauí
Centro de Ciências da Natureza
Programa de Pós-Graduação em Ciência da Computação

Sistema de localização para *Cellbots* integrando odometria visual monocular e força de sinal WiFi

Diego Porto Rocha

Teresina-PI, Setembro de 2016

Diego Porto Rocha

Sistema de localização para *Cellbots* integrando odometria visual monocular e força de sinal WiFi

Diessertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da UFPI (área de concentração: Sistemas de Computação), como parte dos requisitos necessários para a obtenção do Título de Mestre em Ciência da Computação.

Universidade Federal do Piauí – UFPI

Centro de Ciências da Natureza

Programa de Pós-Graduação em Ciência da Computação

Orientador: Dr. André Macedo Santana

Teresina-PI

Setembro de 2016

Aos meus pais Antônia Fernandes Porto e João de Oliveira Rocha,
que sempre me apoiaram nos meus estudos
e sempre estiveram comigo em todos os momentos.

Agradecimentos

Dedico este trabalho primeiramente à Deus, que sempre me deu força e coragem para perseverar em busca da concretização dos meus objetivos.

Aos meus familiares, em especial meus pais, João de Oliveira e Antônia Fernandes, que sempre me apoiaram e aconselharam incentivando a seguir em frente nos momentos mais difíceis.

Agradeço ao meu orientador, Prof. Dr. André Macedo Santana, por dedicar seu tempo, seus conhecimentos, conselhos e pela paciência que ajudou na concretização deste trabalho.

A todos os professores que durante este período dividiram os seus conhecimentos com os futuros educadores.

Aos colegas do laboratório RAPOSA por todos os bons momentos, pizzas, bagunças, força e incentivo no dia-a-dia. Aos meus amigos pela compreensão, torcida e pensamentos positivos.

À Capes/FAPEPI pelo apoio financeiro para realização deste trabalho de pesquisa.

E a todos os demais que contribuíram para esta vitória.

"A ciência não é mais que uma perversão de si mesma,
se o seu objetivo final não for a melhoria da humanidade."
(Nikola Tesla)

Resumo

Um dos principais desafios da robótica móvel consiste em dotar robôs de autonomia. Neste sentido, aflora a necessidade de um sistema capaz de computar a pose (posição e orientação) do robô no seu espaço de trabalho – Sistema de localização. Este trabalho apresenta um sistema de localização utilizando odometria visual, filtro de partículas e sinal *WiFi*. O sistema foi projetado para ser utilizando em uma plataforma robótica da arquitetura *cellbot* (robôs baseados em dispositivos móveis) e faz uso do filtro probabilístico de partículas para realizar a correção da pose do robô no ambiente. Foram realizadas análises de custo computacional e precisão combinando algoritmos de detecção, descrição e correspondência na odometria visual, bem como a influência da quantidade de *features* detectadas no desempenho da odometria. Também foram analisados a precisão e custo da diversificação na quantidade de partículas utilizadas no filtro e a influência da quantidade de pontos de acesso *WiFi* utilizados. Os resultados experimentais com plataforma robótica real e simulação são analisados e apresentados neste trabalho. Um cenário utilizando uma robô cadeira de rodas foi proposto e analisado, o sistema demonstrou ser capaz de localizar o robô móvel.

Palavras-chave: Localização. Robótica móvel. *Cellbots*.

Abstract

One of the most challenges in mobile robotics is to provide robot autonomy. In this sense, arises the need of a system able to compute the pose (position and orientation) of the robot in its workspace – Location System. This paper presents a location system using visual odometry, particle filter and *WiFi* signal. The system is designed to be used on a *cellbot* robotic platform architecture (robots based on mobile devices) and makes use of probabilistic particle filter to perform the pose correction of the robot in the environment. Computational cost and precision analysis combining detection, description, matching algorithms for the visual odometry were performed, and the influence of the amount of *features* detected in the odometry performance. Also the accuracy and cost on diversification of the amount of particles used in the filter and the influence of the number of *WiFi* access points used were analyzed. The experimental results with a real robotic platform and simulation are analyzed and presented in this paper. A scenario using a wheelchair robot was proposed and analyzed, the system demonstrated being able to locate the mobile robot.

Keywords: Localization. Mobile Robotics. Cellbots.

Lista de ilustrações

Figura 1 – Exemplos de tipos de robôs. 1
Figura 2 – Robô móvel Curiosity desenvolvido pela *NASA* para exploração de Marte. 2
Figura 3 – Arquitetura de navegação de um sistema robótico. 2
Figura 4 – Fluxograma básico da localização de robôs móveis. 4
Figura 5 – Exemplificação da robótica probabilística. (THRUN; BURGARD; FOX, 2005). 5
Figura 6 – Visão geral do sistema proposto utilizando *cellbot*. 6
Figura 7 – Ilustração do funcionamento do sistema proposto. 7
Figura 8 – Um modelo de *cellbot*. 7
Figura 9 – Imagem do robô *PewBot I*. 8
Figura 10 – Exemplos do odômetro de Da Vinci. 11
Figura 11 – Posicionamento resultante da integração de deslocamentos ao longo do tempo. 12
Figura 12 – Variáveis do modelo geométrico. 13
Figura 13 – Exemplo do funcionamento da odometria visual. 15
Figura 14 – Fluxo de execução da odometria visual. 16
Figura 15 – Imagens ao lado esquerdo foram capturadas no tempo k e ao lado direito no tempo $k + 1$. a) Sequência de imagens capturadas pela câmera. b) Detecção e descrição das *features*. c)Correspondência das *features* detectadas. 17
Figura 16 – Exemplo do mapa orientado por intensidade de sinal *WiFi* (BISWAS; VELOSO, 2010). 20
Figura 17 – Etapas do filtro de partículas. Adaptado de (THRUN; BURGARD; FOX, 2005) 22
Figura 18 – Fluxo de execução do sistema de localização. 25
Figura 19 – Exemplo do mapa topológico. 28
Figura 20 – Robô sendo simulado na ferramenta V-rep. 29
Figura 21 – a) Robô construído de Lego; b) Interface do programa de localização no computador. 31
Figura 22 – a) Nó sensor, mote; b) Interface do programa de localização no computador. 31
Figura 23 – Imagens utilizadas na análise dos algorítimos para *VO*. 37
Figura 24 – Imagens da base KITTI utilizadas na análise dos algorítimos para *VO*. 38
Figura 25 – Trajetórias utilizando o descritor *BRISK* e o comparador *BruteForce*. . 43
Figura 26 – Trajetórias utilizando o descritor *ORB* e o comparador *BruteForce*. . . 43
Figura 27 – Trajetórias utilizando o descritor *BRISK* e o comparador *BruteForce-Hamming*. 44

Figura 28 – Trajetórias utilizando o descritor *ORB* e o comparador *BruteForce-Hamming*. 44
Figura 29 – Trajetórias utilizando o descritor *BRISK* e o comparador *BruteForce-Hamming LUT*. 45
Figura 30 – Trajetórias utilizando o descritor *ORB* e o comparador *BruteForce-Hamming LUT*. 45
Figura 31 – Trajetórias utilizando o descritor e detector *AKAZE*. 46
Figura 32 – Imagem do ambiente utilizado para os testes do cenário 1. 47
Figura 33 – Comparação da trajetória do robô utilizando uma média de 824 *features*. 48
Figura 34 – Gráfico das médias do erro quadrático da trajetória no primeiro cenário. 48
Figura 35 – Gráfico das médias e desvios do erro quadrático da trajetória no primeiro cenário. 49
Figura 36 – Gráfico das médias e desvios do erro quadrático da trajetória no primeiro cenário. 49
Figura 37 – Gráfico das médias e desvios do erro quadrático da trajetória no primeiro cenário. 50
Figura 38 – Imagem do mapa do Departamento de Computação, ambiente utilizado para os testes do cenário 2. 51
Figura 39 – Comparação da trajetória do robô utilizando 1000 partículas. 51
Figura 40 – Médias para o erro quadrático da trajetória variando a quantidade de partículas. 52
Figura 41 – Médias e desvio dos tempos de execução do filtro variando a quantidade de partículas no smartphone LG Nexus 5. 52
Figura 42 – Médias e desvio dos tempos de execução do filtro variando a quantidade de partículas no smartphone Motorola XT1097. 53
Figura 43 – Médias e desvio dos tempos de execução do filtro variando a quantidade de partículas no smartphone Samsung SM-T520. 53
Figura 44 – Dimensões padrões de uma cadeira de rodas segundo a "NBR9050" ("2015"). 54
Figura 45 – Imagem que demonstra o ambiente do terceiro cenário, uma ala hospitalar. 55
Figura 46 – Comparação da trajetória do robô utilizando 3 *Access Points*. 55
Figura 47 – Médias para o erro quadrático da trajetória variando a quantidade de *APs*. 56
Figura 48 – Comparação da trajetória do robô utilizando uma média de 248 *features*. 67
Figura 49 – Comparação da trajetória do robô utilizando uma média de 630 *features*. 68
Figura 50 – Comparação da trajetória do robô utilizando uma média de 1254 *features*. 68
Figura 51 – Comparação da trajetória do robô utilizando uma média de 2472 *features*. 69
Figura 52 – Comparação da trajetória do robô utilizando uma média de 3064 *features*. 69
Figura 53 – Comparação da trajetória do robô utilizando uma média de 3642 *features*. 70

Figura 54 – Comparação da trajetória do robô utilizando 250 partículas. 71
Figura 55 – Comparação da trajetória do robô utilizando 500 partículas. 71
Figura 56 – Comparação da trajetória do robô utilizando 750 partículas. 72
Figura 57 – Comparação da trajetória do robô utilizando 1250 partículas. 72
Figura 58 – Comparação da trajetória do robô utilizando 1500 partículas. 73
Figura 59 – Comparação da trajetória do robô utilizando 1750 partículas. 73
Figura 60 – Comparação da trajetória do robô utilizando 1 *Access Point*. 74
Figura 61 – Comparação da trajetória do robô utilizando 5 *Access Points*. 74
Figura 62 – Comparação da trajetória do robô utilizando 7 *Access Points*. 75
Figura 63 – Comparação da trajetória do robô utilizando 10 *Access Points*. 75

Lista de tabelas

Tabela 1 – Principais características dos smartphones utilizados no projeto durante a Análise 1. 27
Tabela 2 – Principais características dos smartphones utilizados no projeto durante a Análise 2. 27
Tabela 3 – Tabela com as médias e desvios padrão dos tempos de execução dos algoritmos no Motorola XT1058, utilizando a correspondência por *BruteForce-L1*. 33
Tabela 4 – Tabela com as médias e desvios padrão dos tempos de execução dos algoritmos no Motorola XT1058, utilizando a correspondência por *BruteForce-L2*. 34
Tabela 5 – Tabela com as médias e desvios padrão dos tempos de execução dos algoritmos no Motorola XT 1058, utilizando a correspondência por *BruteForce-Hamming* 34
Tabela 6 – Tabela com as médias e desvios padrão dos tempos de execução dos algoritmos no Motorola XT1097, utilizando a correspondência por *BruteForce-L1*. 34
Tabela 7 – Tabela com as médias e desvios padrão dos tempos de execução dos algoritmos no Motorola XT1097, utilizando a correspondência por *BruteForce-L2*. 35
Tabela 8 – Tabela com as médias e desvios padrão dos tempos de execução dos algoritmos no Motorola XT1097, utilizando a correspondência por *BruteForce-Hamming* 35
Tabela 9 – Tabela com as médias e desvios padrão dos tempos de execução dos algoritmos no Nexus 5, utilizando a correspondência por *BruteForce-L1*. 36
Tabela 10 – Tabela com as médias e desvios padrão dos tempos de execução dos algoritmos no Nexus 5, utilizando a correspondência por *BruteForce-L2*. 36
Tabela 11 – Tabela com as médias e desvios padrão dos tempos de execução dos algoritmos no Nexus 5, utilizando a correspondência por *BruteForce-Hamming* 36
Tabela 12 – Tabela com as médias e desvios padrão em milissegundos dos tempos de execução dos algoritmos no Samsung SM-T520, utilizando a correspondência por *BruteForce* 38
Tabela 13 – Tabela com as médias e desvios padrão em milissegundos dos tempos de execução dos algoritmos no Samsung SM-T520, utilizando a correspondência por *BruteForc-Hamminge* 38

Tabela 14 – Tabela com as médias e desvios padrão em milissegundos dos tempos de execução dos algoritmos no Samsung SM-T520, utilizando a correspondência por *BruteForce-Hamming Look-up Table* 39
Tabela 15 – Tabela com as médias e desvios padrão em milissegundos dos tempos de execução dos algoritmos no LG Nexus 5, utilizando a correspondência por *BruteForce* 39
Tabela 16 – Tabela com as médias e desvios padrão em milissegundos dos tempos de execução dos algoritmos no LG Nexus 5, utilizando a correspondência por *BruteForce-Hamming* 39
Tabela 17 – Tabela com as médias e desvios padrão em milissegundos dos tempos de execução dos algoritmos no LG Nexus 5, utilizando a correspondência por *BruteForce-Hamming Look-up Table* 40
Tabela 18 – Tabela com as médias e desvios padrão em milissegundos dos tempos de execução dos algoritmos no Motorola XT1097, utilizando a correspondência por *BruteForce* 40
Tabela 19 – Tabela com as médias e desvios padrão em milissegundos dos tempos de execução dos algoritmos no Motorola XT1097, utilizando a correspondência por *BruteForce-Hamming* 41
Tabela 20 – Tabela com as médias e desvios padrão em milissegundos dos tempos de execução dos algoritmos no Motorola XT1097, utilizando a correspondência por *BruteForce-Hamming Look-up Table* 41
Tabela 21 – Tabela com a quantidade média de *features* capturadas durante a execução dos algoritmos nos smartphones 41
Tabela 22 – Combinações mais rápidas de cada detector com seus tempos de execução, erros e desvio padrão em relação à pose real 42
Tabela 23 – Combinações mais precisas com seus tempos de execução, erros e desvio padrão em relação à pose real 46
Tabela 24 – Tabela com as médias e desvios padrão em milissegundos dos tempos de execução dos algoritmos no Computador, utilizando a correspondência por *BruteForce* 65
Tabela 25 – Tabela com as médias e desvios padrão em milissegundos dos tempos de execução dos algoritmos no Computador, utilizando a correspondência por *BruteForce-Hamming* 65
Tabela 26 – Tabela com as médias e desvios padrão em milissegundos dos tempos de execução dos algoritmos no Computador, utilizando a correspondência por *BruteForce-Hamming Look-up Table* 66

Lista de abreviaturas e siglas

2D	Bidimensional
3D	Tridimensional
a.C.	Antes de Cristo
AGAST	*Adaptive and Generic Accelerated Segment Test*
AKAZE	*Accelerated-KAZE*
AP	*Access Point*
API	*Application Programming Interface*
BF	*BruteForce*
BFH	*BruteForce-Hamming*
BRIEF	*Binary Robust Independent Elementary Features*
BRISK	*Binary Robust Invariant Scalable Keypoints*
CDC	*Communications Device Class*
CPU	*Central Processing Unit*
dBm	*decibel miliwatt*
DC	*Departamento de Computação*
EEM	*Ethernet Emulation Model*
FAST	*Features from Accelerated Segment Test*
FREAK	*Fast Retina Keypoint*
GB	*Gigabyte*
GFTT	*Good Features To Track*
GHz	*Gigahertz*
GPS	*Global Positioning System*
HU	Hospital Universitário

ICIAP	*International Conference on Image Analysis and Processing*
IDE	*Integrated Development Environment*
IEEE	*Institute of Electrical and Electronic Engineers*
IMU	*Inertial Measurement Unit*
JNI	*Java Native Interface*
LIDAR	*Light Detection And Ranging*
LUT	*Look-up Table*
m	metros
ms	milissegundos
ML	*Machine Learning*
MSER	*Maximally Stable Extremal Regions*
NASA	*National Aeronautics and Space Administration*
NDK	*Native Development Kit*
ODE	*Open Dynamics Engine*
OpenCV	*Open source Computer Vision library*
ORB	*Oriented FAST and Rotated BRIEF*
RAM	*Random Access Memory*
RSP	*Reflection Signal Phase*
RSS	*Received Signal Strength*
RTOF	*Reflection Time Of Flight*
SDK	*Software Development Kit*
SIFT	*Scale-Invariant Feature Transform*
SLAM	*Simultaneous Localization and Mapping*
SO	Sistema Operacional
SURF	*Speeded-Up Robust Features*
TDOA	*Time Difference Of Arrival*

TOA	*Time Of Arrival*
USB	*Universal Serial Bus*
VO	*Visual Odometry*
WLAN	*Wireless Local Area Network*

Lista de símbolos

x	Posição do robô na coordenada cartesiana x
y	Posição do robô na coordenada cartesiana y
Θ	Ângulo que representa a orientação do robô
ΔL	Deslocamento linear do robô
$\Delta\Theta$	Deslocamento angular do robô
x_t	Posição do robô na coordenada x no tempo t
y_t	Posição do robô na coordenada y no tempo t
Θ_t	Ângulo do robô no tempo t
b	Comprimento do eixo do robô
r_d	Raio da roda direita do robô
r_e	Raio da roda esquerda do robô
ε_d	Ruído gaussiano da roda direita
ε_e	Ruído gaussiano da roda esquerda
C_k	Matriz de transformações da câmera no tempo k com relação ao ponto inicial
$T_{k,k-1}$	Matriz de transformações entre o instante anterior $(k-1)$ e o instante atual (k)
$z_{1:t}$	Conjunto de observações sensoriais realizados pelo robô
$u_{1:t}$	Conjunto de entradas de controle aplicadas no robô
η	Valor de normalização do peso das partículas no filtro
X_t	Conjunto de partículas representado os estados do robô
^{M}	Número total de amostras

Sumário

1 INTRODUÇÃO 1
1.1 O problema de Localização 3
1.2 Visão Geral da Proposta 6
1.2.1 *CellBots* 7
1.3 Motivação e Justificativa 8
1.4 Objetivos 9
1.5 Organização dos Capítulos 9

2 MATERIAIS E MÉTODOS 11
2.1 Odometria Mecânica 11
2.2 Odometria Visual 14
2.3 Localização *indoor* com *WiFi* 18
2.4 Filtro de Bayes 20
2.5 Filtro de Partículas 20
2.6 Android 22
2.7 *OpenCV* 23
2.8 Considerações Finais 23

3 SISTEMA PROPOSTO 25
3.1 Simulador V-rep 28
3.2 Trabalhos de Influência 30
3.3 Considerações Finais 31

4 RESULTADOS E DISCUSSÃO 33
4.1 Análise 1 33
4.2 Análise 2 37
4.3 Cenários de Localização 46
4.3.1 Cenário 1 47
4.3.2 Cenário 2 47
4.3.3 Cenário 3 54

5 CONCLUSÕES E TRABALHOS FUTUROS 57

REFERÊNCIAS 59

APÊNDICES 63

APÊNDICE A – TEMPOS DE EXECUÇÃO NO COMPUTADOR . 65

APÊNDICE B – TRAJETÓRIAS DO ROBÔ COM RELAÇÃO A TRAJETÓRIA REAL 67

B.1 Quanto a variação da quantidade de *features* 67

B.2 Quanto a variação da quantidade de partículas 71

B.3 Quanto a variação da quantidade de *APs* no ambiente 74

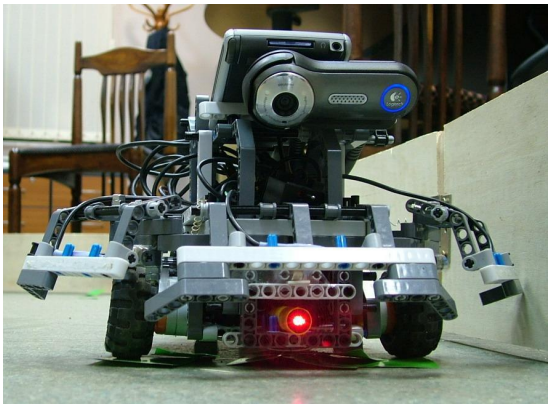
1 Introdução

A robótica tem alcançado grandes conquistas no mundo atual e na indústria. Os braços robóticos (manipuladores), por exemplo, conseguem realizar tarefas repetitivas com grande velocidade e precisão em uma linha de montagem (Figura 1a). Na indústria eletrônica, os braços robóticos realizam a montagem de componentes com precisão sobre-humana, tornando possível a criação de dispositivos de *hardware* com maior complexidade e tamanho reduzido.

Mesmo com todo esse sucesso, esses robôs sofrem da falta de mobilidade, o que lhes representa uma desvantagem. Um braço manipulador fixo possui um alcance e movimentação limitados. Em contrapartida, a robótica móvel tem como principal característica a autonomia dos robôs, que lhes permitem deslocar-se pelo ambiente em que estão, como ilustra na Figura 1b.



(a) Robô manipulador industrial.



(b) Robô móvel.

Figura 1 – Exemplos de tipos de robôs.

Craig (2005) disse que os robôs são classificados em dois grandes grupos: os manipuladores e os móveis. O grupo dos robôs móveis, por sua vez, pode ser dividido em: robôs aeroespaciais (deslocam-se no espaço tridimensional) ou terrestres (deslocam-se no espaço planar). Os robôs terrestres, por sua vez, podem ser agrupados em: robôs com rodas, robôs rastejantes e robôs humanoides.

Em ambientes hostis têm-se popularizado o uso de robôs moveis teleoperados, por exemplo: robôs policiais que desarmam bombas e robôs exploradores em acidentes nucleares. Porém, algumas vezes a situação se torna complexa ao ponto de que o robô não pode ser controlado diretamente pelo teleoperador e necessita navegar por conta própria como, por exemplo, os robôs enviados em missões de exploração espacial (Figura 2).

A navegação é um dos campos de estudo mais desafiadores da robótica. Alsina

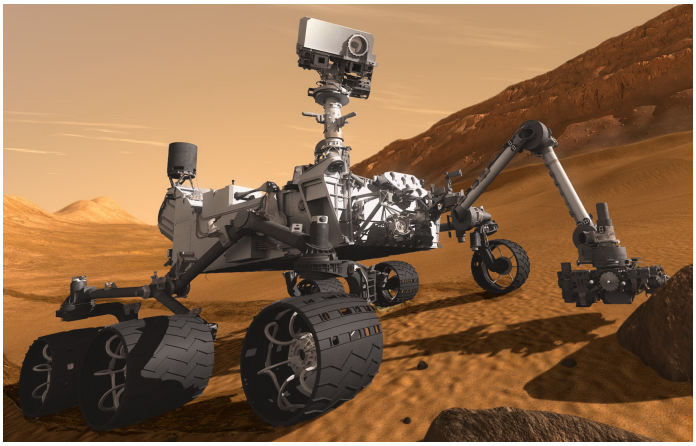


Figura 2 – Robô móvel Curiosity desenvolvido pela *NASA* para exploração de Marte.

(SANTANA, 2007 apud ALSINA et al., 2002) divide o processo de navegação em cinco níveis hierárquicos: modelagem do ambiente, localização, geração de caminho, geração de trajetória e execução da trajetória. A Figura 3 ilustra a arquitetura.

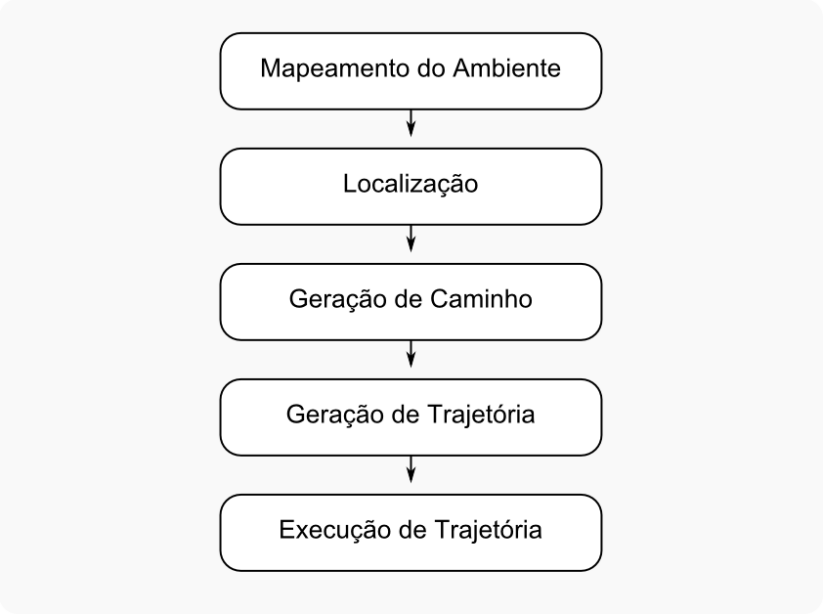


Figura 3 – Arquitetura de navegação de um sistema robótico.

- **Modelagem do ambiente**: neste nível o robô utiliza sensores para coletar informações e criar um modelo com as principais características estruturais do ambiente.
- **Localização**: é determinada a posição e orientação do robô no espaço, com base no modelo obtido no nível de modelagem do ambiente.
- **Geração de caminho**: o robô calcula a curva geométrica que o leva de sua localização inicial à sua localização final, desviando dos obstáculos, quando existirem.
- **Geração de trajetória**: o caminho calculado no nível anterior é adaptado às restrições temporais impostas. A partir daí são calculadas as velocidades que devem

ser aplicadas ao mesmo para que possa percorrer o caminho no intervalo de tempo especificado.

- **Execução de trajetória**: os atuadores do robô são controlados de forma que a trajetória percorrida por ele seja a mais próxima possível da calculada no nível anterior.

Dentre os cinco níveis hierárquicos que compõem o processo de navegação, este trabalho tratará especificamente do processo de localização. O problema de localização na robótica móvel consiste em, a partir de leituras realizadas pelos sensores do robô e um conhecimento prévio das características do ambiente (mapa), o robô ser capaz de determinar e manter atualizada sua pose (posição e orientação) com relação ao ambiente que se encontra, ainda que existam erros ou ruídos nas leituras.

1.1 O problema de Localização

Se fosse viável adicionar um sensor de *GPS* (*Global Positioning System*) a um robô, grande parte do problema de localização estaria resolvido. O *GPS* é um sistema de navegação utilizando satélites que fornece a um dispositivo receptor móvel a sua posição, assim como informação horária. O *GPS* atua independente das condições atmosféricas e em qualquer lugar na Terra, desde que o receptor se encontre no campo de visão de quatro satélites *GPS*.

Os dispositivos com sensor *GPS* da atualidade possuem uma precisão de cerca de alguns metros e para a localização *indoor* o nível da escala dos resultados produzidos pelo *GPS* se torna inviável. A localização *indoor* muitas vezes necessita de um nível de resolução com uma escala menor que a produzida pelo *GPS*. Além disso, a tecnologia do *GPS* não opera bem em locais fechados ou áreas obstruídas, limitando sua área de trabalho. Assim, é necessário a criação de estratégias que auxiliem o robô a se encontrar em um determinado ambiente.

Borenstein e Feng (1996) citam que os métodos utilizados para a localização podem ser agrupados em dois tipos: localização com posicionamento relativo e localização com posicionamento absoluto. A localização com posicionamento relativo é simples, barata e rápida. Ela faz estimativa da pose do robô com relação ao seu ponto de partida (por exemplo, o uso da odometria). Porém, este método está sujeito ao acúmulo de erros. A localização com posicionamento absoluto, por outro lado, faz a estimativa da pose global do robô, ou seja, a pose com relação ao ambiente em que se encontra (por exemplo, o sensor de *GPS*). Este método é mais complexo, lento e geralmente o custo para ser implantado é maior.

A Figura 4 ilustra o conceito básico de localização. O processo de localização se divide em duas etapas: predição e percepção. Inicialmente, na etapa de predição, o prognóstico da pose é feito através de uma estimativa do deslocamento realizado pelo robô no ambiente (posicionamento relativo), obtido através de uma leitura do *encoder*. Uma vez que a pose foi predita, é feita uma tentativa de correspondência dela junto com as leituras dos sensores (posicionamento absoluto) e as informações do mapa. Caso haja uma correspondência, então é possível fazer a localização do robô naquele momento e uma atualização da pose é feita. Ao final, o processo é repetido.

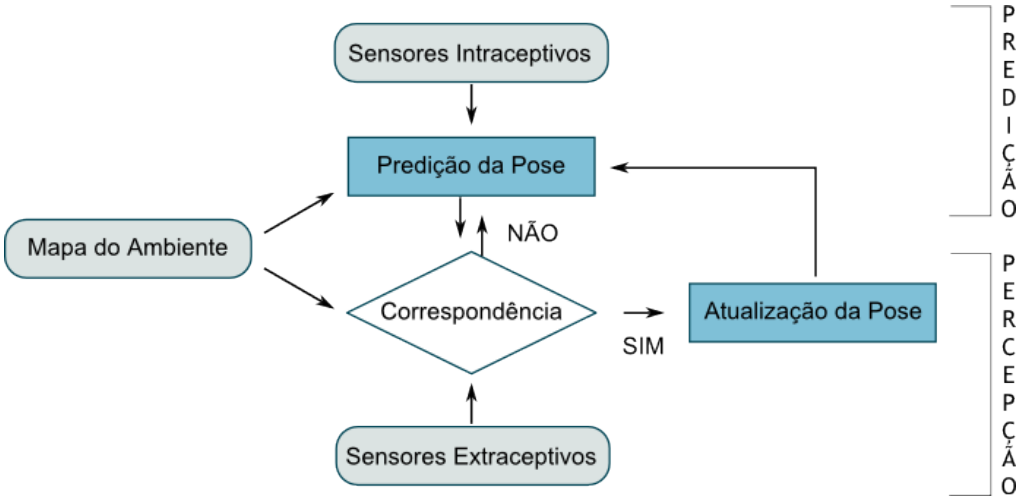


Figura 4 – Fluxograma básico da localização de robôs móveis.

Os sensores para um robô são seu principal meio de perceber o ambiente ao seu redor. Thrun, Burgard e Fox (2005) descrevem o ruído de um sensor como sendo uma inconsistência que acontece de forma imprevisível nas leituras das informações e, assim, limita os dados que podem ser extraídos durante as leituras. O acúmulo de ruído gera uma localização errônea, fazendo com que se consiga somente computar a probabilidade de um robô estar em uma determinada configuração. Dessa forma, uma abordagem conhecida por robótica probabilística (THRUN; BURGARD; FOX, 2005) é utilizada para representar a localização do robô. Esta abordagem consiste em representar uma determinada configuração do robô através de uma distribuição probabilística sobre todas as suas possíveis poses. Assim, a ambiguidade e o grau de confiança são tratados como cálculos da teoria probabilística.

A Figura 5 demonstra o funcionamento da robótica probabilística, onde o robô se desloca pelo ambiente realizando leituras sensoriais. O $bel(x)$ representa o *belief* (confiança) daquele estado (probabilidade de o robô estar em uma determinada posição x) e o $p(z|x)$ é a probabilidade do robô estar no estado x dada uma medição do sensor z. Considerando um posicionamento unidimensional ao longo do eixo das abscissas e o valor da confiança de uma probabilidade no eixo das ordenadas, observa-se que em:

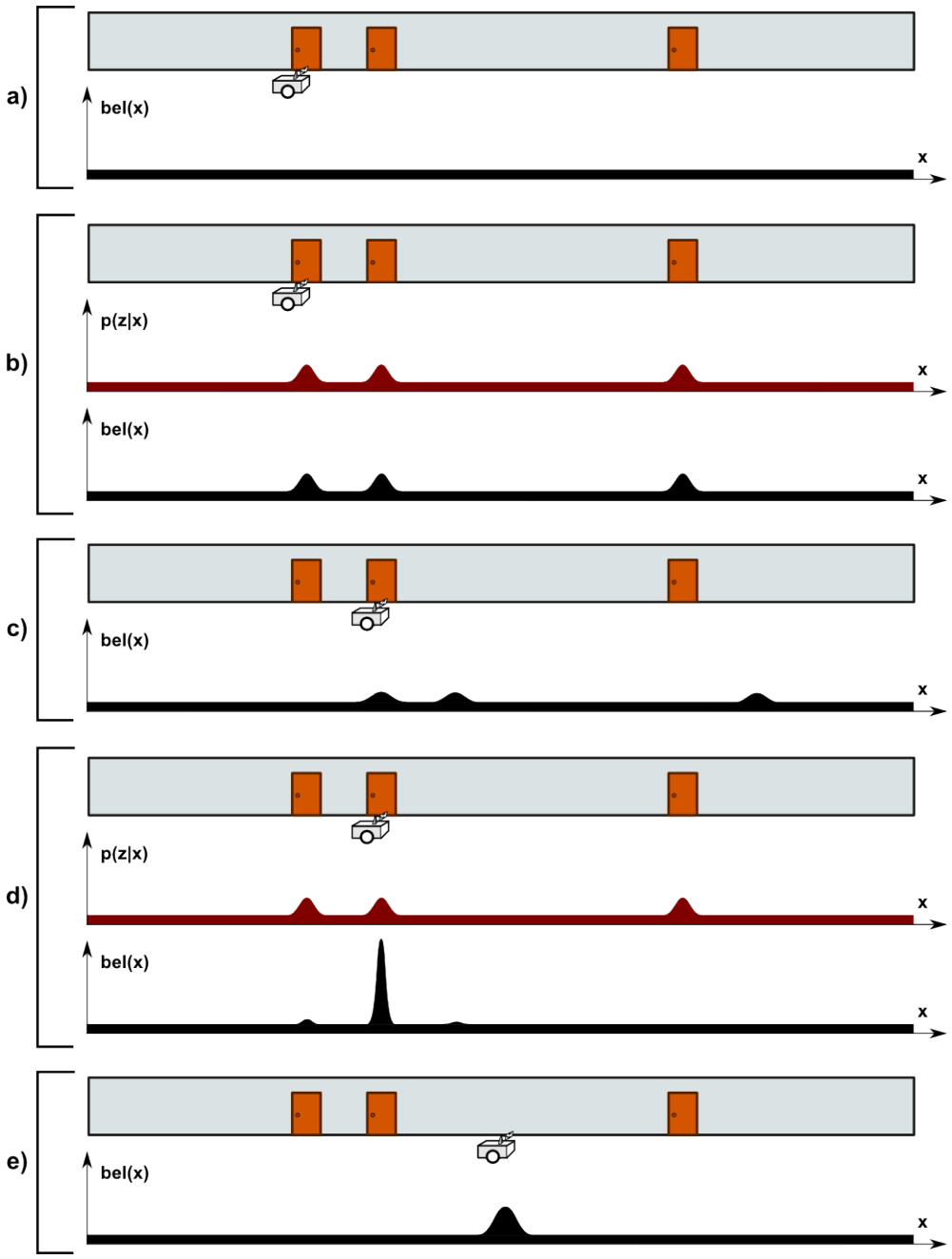


Figura 5 – Exemplificação da robótica probabilística. (THRUN; BURGARD; FOX, 2005).

- **a)** O robô encontra-se em seu estado inicial, a possível posição do robô é indeterminada.

- **b)** Uma leitura sensorial do ambiente é realizada, elevando a probabilidade do robô estar em uma posição próxima a uma porta. As possíveis posições do robô são estimadas.

- **c)** O robô realiza um deslocamento e a incerteza de sua posição aumenta.

- **d)** Mais uma vez, uma leitura sensorial é realizada e a confiança da posição do robô começa a convergir para um único local.

- **e)** O robô realiza mais um deslocamento e sua possível posição está determinada.

1.2 Visão Geral da Proposta

Este trabalho apresenta um sistema de localização para robôs utilizando um smartphone. O sistema utilizará como sensores a câmera, o compasso digital e o receptor *WiFi* do smartphone (Figura 6).

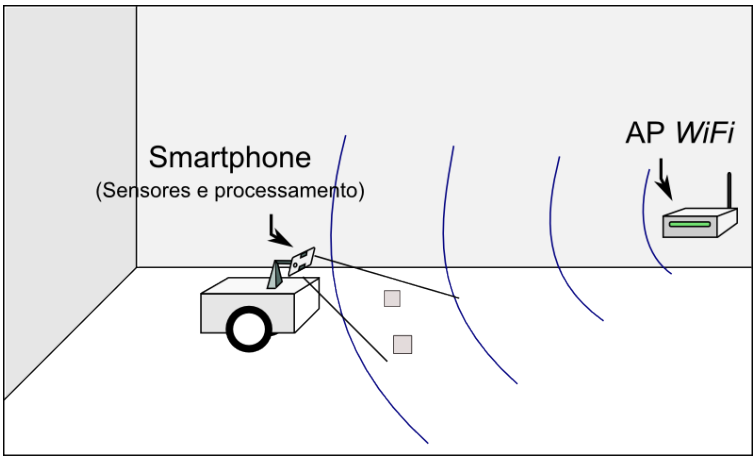


Figura 6 – Visão geral do sistema proposto utilizando *cellbot*.

Esse sistema irá realizar odometria visual, a partir dos quadros de imagem obtidos pela câmera, para estimar seu deslocamento e utilizará a força de sinal *WiFi* de *APs* (*Access Points*) junto com um filtro de partículas para corrigir sua posição no ambiente. Assim, o sistema é capaz de informar a pose do robô dentro de um determinado ambiente durante seu deslocamento (Figura 7).

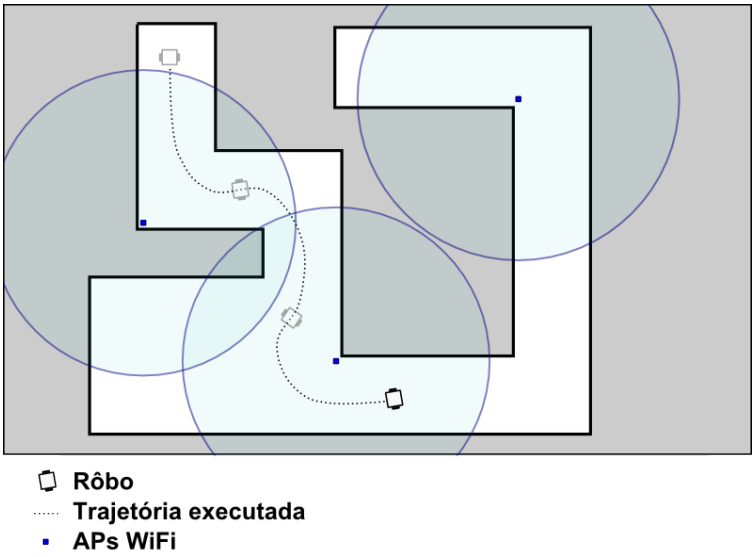


Figura 7 – Ilustração do funcionamento do sistema proposto.

1.2.1 *CellBots*

Devido ao alto custo do hardware de um robô móvel, propostas como o *cellbot* e arquiteturas robóticas de baixo custo se tornam cada vez mais comuns. O termo *cellbot* é uma proposta de arquitetura para sistemas robóticos móveis que faz a utilização de um dispositivo móvel (smartphone) ou tablet que permita o desenvolvimento de aplicações, como uma unidade de comando para os sensores e atuadores de um robô, como descrito por Aroca et al. (2013) e ilustrado na Figura 8.



Figura 8 – Um modelo de *cellbot*.

Este modelo de arquitetura permite que o robô tire vantagem dos sensores e características dos smartphones, em sua maior parte equipados com um sensor do sistema de posicionamento global (*global positioning system*, *GPS*), câmera, bússola, acelerômetro, módulo de comunicação *WiFi*, módulo de comunicação *Bluetooth*, saídas de som, microfone e muitas outras características que encaixam perfeitamente no uso da robótica móvel.

Santos, Tarrataca e Cardoso (2010) provou ser viável o desenvolvimento e execução de algoritmos de navegação para robôs móveis utilizando um telefone móvel. Nesta abordagem a implementação do algoritmos realizada por Santos et al. demostrou alta consistência e robustez em sua análise. Considerando as vantagens citadas, optou-se pela utilização da arquitetura *cellbot* neste trabalho.

1.3 Motivação e Justificativa

Os processos de localização e navegação são de grande importância para a realização das tarefas de um robô móvel. Atualmente, uma variedade de abordagens diferentes com soluções para o problema de localização tem sido exploradas. Porém, a localização *indoor* ainda é uma tarefa desafiadora.

A principal motivação deste trabalho é o desenvolvimento de um sistema para o problema de localização que seja eficiente utilizando um hardware acessível e de baixo custo. Isto será obtido utilizando-se um robô com rodas conectado a um smartphone, onde o smartphone será responsável por realizar o sensoriamento e processamento da informação necessária para a localização do robô móvel que será apresentado.

O sistema a ser desenvolvido irá operar em um smartphone da plataforma Android em conjunto com um pequeno *hardware* embarcado que ficará responsável pelo acionamento dos motores do robô. O robô, chamado de *PewBot I*, possui dois motores de corrente contínua com *encoders* acoplados no eixo de cada roda. A Figura 9 ilustra o robô utilizado neste trabalho.

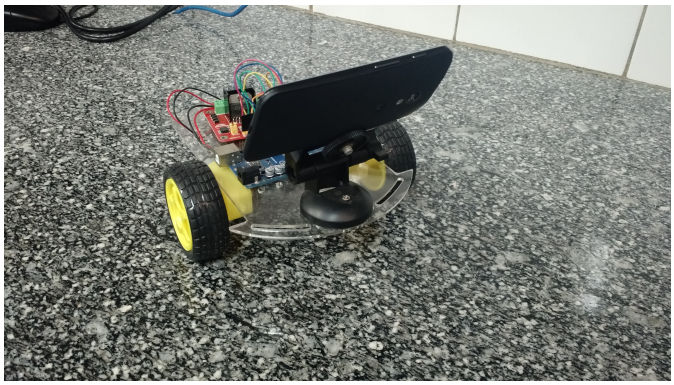


Figura 9 – Imagem do robô *PewBot I*.

Atualmente existem muitos robôs comerciais disponíveis para o desenvolvimento de pesquisa científica. Os robôs comerciais mais populares custam na faixa de 4.500,00 dólares. A utilização de um *cellbot* propõe reduzir os custos financeiros na elaboração de uma aplicação robótica. Assim, pretende-se analisar o desempenho e o benefício da execução de um sistema de localização *indoor* em um smartphone. Este projeto criará um

sistema capaz de rastrear a posição e orientação de uma robô cadeira de rodas em um ambiente hospitalar.

1.4 Objetivos

Objetivo Geral: O objetivo deste trabalho é o desenvolvimento de um sistema de localização para um robô móvel da arquitetura *cellbot* utilizando um robô com rodas e um smartphone.

Objetivos Específicos:

- desenvolver um sistema de localização eficiente para *cellbots* utilizando odometria visual e força de sinal *WiFi*;
- analisar o comportamento, eficiência e viabilidade dos algoritmos de detecção de *features* para odometria visual nos *cellbots*;
- analisar o comportamento, eficiência e viabilidade da localização utilizando sinal *WiFi* em *cellbots*;
- analisar o comportamento, eficiência e viabilidade dos algoritmos de localização clássicos da literatura nos *cellbots*;
- analisar a viabilidade da utilização do sistema para a localização de uma cadeira de rodas robô em um ambiente hospitalar.

1.5 Organização dos Capítulos

O trabalho encontra-se dividido em cinco capítulos. Neste primeiro capítulo foi feita a introdução ao trabalho e apresentadas as suas principais características e propósito. O capítulo 2 apresenta as principais ferramentas computacionais utilizadas ao longdo deste trabalho. O capítulo 3 descreve com detalhes a proposta deste trabalho para a criação do sistema de localização. No capítulo 4 são apresentados os resultados obtidos a partir de análises feitas no sistema proposto. Por fim, as conclusões sobre os resultados, trabalhos futuros e produção científica gerada são apresentados no capítulo 5.

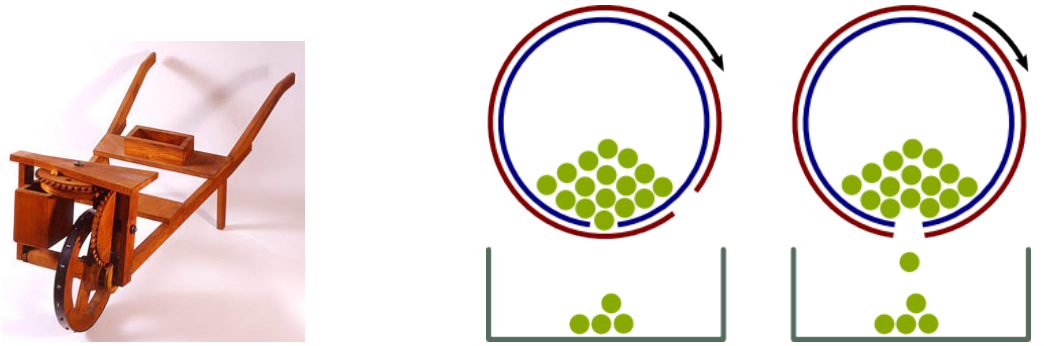
2 Materiais e Métodos

Neste capítulo serão abordadas as principais ferramentas matemáticas e métodos computacionais utilizados para a produção deste trabalho, como odometria mecânica; odometria visual; métodos de localização com sinal *WiFi*; Filtros probabilísticos; sistema Android e a biblioteca OpenCV.

2.1 Odometria Mecânica

A ação de movimento do robô é tradicionalmente descrita pelo seu modelo dinâmico. A modelagem deste tipo de sistema é bastante complexa pois está diretamente ligada a um conjunto de características individuais do robô tais como dimensões, massas e momentos de inércia, atritos, atuadores e etc. Para isso, diversos métodos e estratégias são utilizados para que se possa retratar o movimento realizado da maneira mais próxima à realidade possível.

O conceito inicial da odometria mecânica foi elaborado por Vitruvius no século I a.C (SLEESWYK, 1979). Por volta do ano 1500, Leonardo da Vinci desenvolveu um aparato que era composto por tambores e utilizava pedras pra calcular a distância percorrida. O aparato era composto de um contentor, fixo à estrutura do veículo, cheio de pedras e com uma abertura na parte inferior. A rotação de uma das rodas fazia rodar um tambor em torno do contentor e o tambor possuía uma abertura do mesmo tamanho da existente no contentor. Depois de um certo número de rotações da roda, as duas aberturas coincidem e uma pedra cai, sendo recolhida em uma caixa. O número de pedras recolhidas na caixa em um determinado intervalo de tempo permitia estimar a distância percorrida. Figura 10.



(a) Visão exterior. (b) Representação do funcionamento do aparato.

Figura 10 – Exemplos do odômetro de Da Vinci.

O método da odometria mecânica consiste na determinação da posição e orientação de um robô através da integração dos deslocamentos incrementais de suas rodas, medidos

a partir de um referencial fixo. Para exemplificar esse método, considere um robô se deslocando no seu ambiente seguindo uma trajetória reta, como mostrado na Figura 11. Percebe-se então que a posição atual do robô pode ser calculada baseada no acúmulo dos deslocamentos realizados por ele, com relação a sua posição inicial.

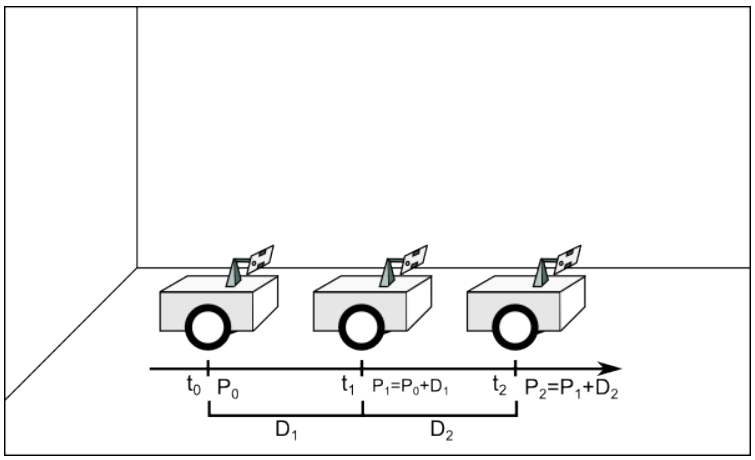


Figura 11 – Posicionamento resultante da integração de deslocamentos ao longo do tempo.

Para medir o deslocamento do robô é necessário a utilização de um sensor que possa medir a rotação das suas rodas. Em aplicações envolvendo robôs móveis, um tipo de sensor de odometria bastante utilizado é o *encoder* óptico. Esse tipo de sensor funciona com base na transmissão e recepção de luz através de um disco perfurado, que gira acoplado ao eixo que movimenta a roda do robô.

Mesmo os sistemas mais refinados não conseguem retratar com fidelidade o comportamento da movimentação dos robôs. A técnica de odometria está sujeita a erros que fazem com que ela forneça apenas uma estimativa da localização exata do robô em um dado instante de tempo (LATOMBE, 1991).

Thrun, Burgard e Fox (2005) propõem que as informações obtidas na odometria não funcionem como medidas sensoriais e sim que elas sejam incorporadas ao modelo do robô. Este modelo utiliza as leituras dos sensores odométricos (*encoders*, imagens e sinal *WiFi*) funcionando como sinais de controle aplicados ao modelo do robô.

Assim, os sinais de controle submetidos aos atuadores do robô não serão valores de tensão, mas sim deslocamentos angulares nas rodas do robô, conforme a Figura 12. Supondo que a velocidade durante os períodos de deslocamentos são constantes, pode-se determinar o modelo geométrico de movimentação do robô, como pode ser visto no Sistema de Equações 2.1:

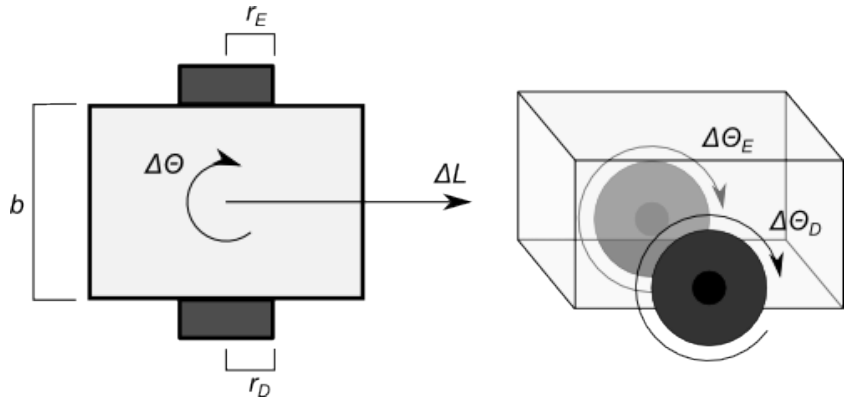


Figura 12 – Variáveis do modelo geométrico.

$$\begin{cases} x_t = x_{t-1} + \dfrac{\Delta L}{\Delta\theta}[\sin(\theta_{t-1} + \Delta\theta) - \sin(\theta_{t-1})] \\ y_t = y_{t-1} + \dfrac{\Delta L}{\Delta\theta}[\cos(\theta_{t-1} + \Delta\theta) - \cos(\theta_{t-1})] \\ \theta_t = \theta_{t-1} + \Delta\theta, \end{cases} \tag{2.1}$$

Onde ΔL corresponde ao deslocamento linear do robô, $\Delta\theta$ corresponde ao deslocamento angular do robô, b ao comprimento do eixo do robô, $\Delta\theta_d$ corresponde ao deslocamento angular da roda direita do robô, $\Delta\theta_e$ corresponde ao deslocamento angular da roda esquerda do robô, r_d corresponde ao raio da roda direita do robô e r_e corresponde ao raio da roda esquerda do robô. As variáveis auxiliares ΔL e $\Delta\theta$ correspondem aos valores do Sistema de Equações 2.2.

$$\begin{cases} \Delta L = (\Delta\theta_d r_d + \Delta\theta_e r_e)/2 \\ \Delta\theta = (\Delta\theta_d r_d - \Delta\theta_e r_e)/b, \end{cases} \tag{2.2}$$

Onde $\Delta\theta_d$ corresponde ao valor da leitura do *encoder* direito e representa o deslocamento angular aplicado à roda direita, $\Delta\theta_e$ corresponde ao valor da leitura do *encoder* esquerdo e representa o deslocamento angular aplicado à roda esquerda, b corresponde à distância do eixo das rodas do robô e r_d e r_e representam os respectivos raios das rodas direita e esquerda do robô.

Porém os valores reais de deslocamento angular realizado pelas rodas do robô diferem do valor obtido na leitura dos *encoders* e a suposição de que as velocidades durante o período de deslocamento serão constantes nem sempre acontece na prática. Assim, essa diferença entre o valor real dos deslocamento angulares ($\Delta\hat{\theta}_d$ e $\Delta\hat{\theta}_e$) e os valores mensurados nos *encoders* ($\Delta\theta_d$ e $\Delta\theta_e$) é modelada como um ruído gaussiano aditivo de acordo com o

Sistema de Equações 2.3,

$$\begin{cases} \Delta\hat{\theta}_d = (\Delta\theta_d + \varepsilon_d) \\ \Delta\hat{\theta}_e = (\Delta\theta_e + \varepsilon_e), \end{cases} \tag{2.3}$$

Onde (ε_d e ε_e) são, respectivamente, o ruído gaussiano da roda direita e esquerda.

$$\begin{cases} \Delta\hat{L} = (\Delta\hat{\theta}_d r_d + \Delta\hat{\theta}_e r_e)/2 \\ \Delta\hat{\theta} = (\Delta\hat{\theta}_d r_d - \Delta\hat{\theta}_e r_e)/b \end{cases} \tag{2.4}$$

Dado que os ruídos ε_d e ε_e não têm variância constante e que o erro odométrico é cumulativo, assume-se que estes ruídos têm desvio padrão proporcional ao módulo do deslocamento mensurado. Adicionando os dados do ruído aos valores do deslocamento angular do Sistema de Equações 2.2, resultará no sistema 2.4. Substituindo ΔL e $\Delta\theta$ do Sistema de Equações 2.1 por $\Delta\hat{L}$ e $\Delta\hat{\theta}$ do Sistema de Equações 2.4 chegamos ao modelo geométrico representado pelo Sistema de Equações 2.5:

$$\begin{cases} x_t = x_{t-1} + \dfrac{\Delta\hat{L}}{\Delta\hat{\theta}}[\sin(\theta_{t-1} + \Delta\hat{\theta}) - \sin(\theta_{t-1})] \\ y_t = y_{t-1} + \dfrac{\Delta\hat{L}}{\Delta\hat{\theta}}[\cos(\theta_{t-1} + \Delta\hat{\theta}) - \cos(\theta_{t-1})] \\ \theta_t = \theta_{t-1} + \Delta\theta \end{cases} \tag{2.5}$$

Assim, a nova pose poderá ser obtida aplicando os deslocamentos angulares de cada roda ao sistema 2.5. Os valores de x_t , y_t, e θ_t correspondem respectivamente às coordenadas cartesianas x, y, e ângulo θ em que o robô se encontra no tempo t com relação à sua pose inicial.

2.2 Odometria Visual

A Odometria Visual (*Visual Odometry*, *VO*) consiste em um processo de estimar o *egomotion* de um determinado robô utilizando somente imagens capturadas através de uma câmera. *Egomotion* é definido como sendo o movimento tridimensional realizado por uma câmera em um ambiente. O domínio de aplicações da odometria visual inclui a robótica, computação vestível, realidade aumentada e setor automotivo.

O termo *VO* foi proposto por Nistér, Naroditsky e Bergen (2004) e escolhido por sua similaridade à odometria mecânica. Neste trabalho foi proposto um sistema que estimava a posição do robô utilizando uma câmera monocular ou uma câmera estéreo. O sistema opera de maneira incremental através da estimativa da pose do veículo utilizando correspondência de características entre imagens do fluxo de quadros obtido pela câmera.

Assim como a odometria mecânica, a *VO* opera de forma incremental estimando a pose através da integração dos deslocamentos realizados pelas *features* (características) detectadas na imagem e assim infere o deslocamento realizado pela câmera. Uma *feature* representa um pedaço de informação que é relevante para a realização de uma determinada tarefa. Neste caso, as *features* irão representar, em imagens diferentes, um mesmo objeto. Para essa detecção é necessária a existência de duas ou mais imagens distintas, obtidas em tempos diferentes durante o deslocamento, onde existam em ambas a detecção da mesma *feature* para que se faça o cálculo do deslocamento realizado.

A Figura 13 demonstra simplificadamente o funcionamento da *VO*, onde cada posição C_k corresponde às transformações da câmera no tempo k com relação ao ponto inicial ($k = 0$) e $T_{k,k-1}$ corresponde ao deslocamento realizado entre o instante anterior ($k - 1$) e o instante atual (k).

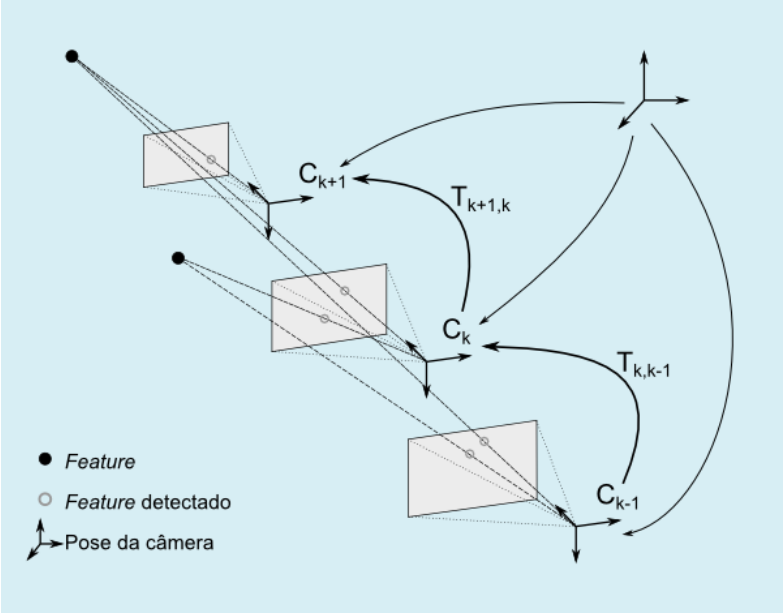


Figura 13 – Exemplo do funcionamento da odometria visual.

Scaramuzza e Fraundorfer (2011) citam o seguinte fluxo, ilustrado na Figura 14, como os principais passos para se realizar odometria visual.

A etapa de sequência de imagens é definida como o conjunto das imagens capturadas pela câmera em diferentes instantes ao longo do deslocamento (Figura 15a). A detecção de *features* significa submeter uma imagem a determinados algoritmos com o objetivo de extrair informações particulares dentro da imagem, como cantos, bordas ou manchas (Figura 15b). Uma vez detectada a *feature*, a região ao redor dela é convertida para um descritor (que contém as informações necessárias para que se realize a correspondência). Após a obtenção dos descritores de cada imagem, o próximo passo é realizar a correspondência entre eles. Na correspondência de *features*, as informações dos descritores gerados na primeira imagem serão comparadas com a segunda imagem. Caso a informação dos descritores se iguale,

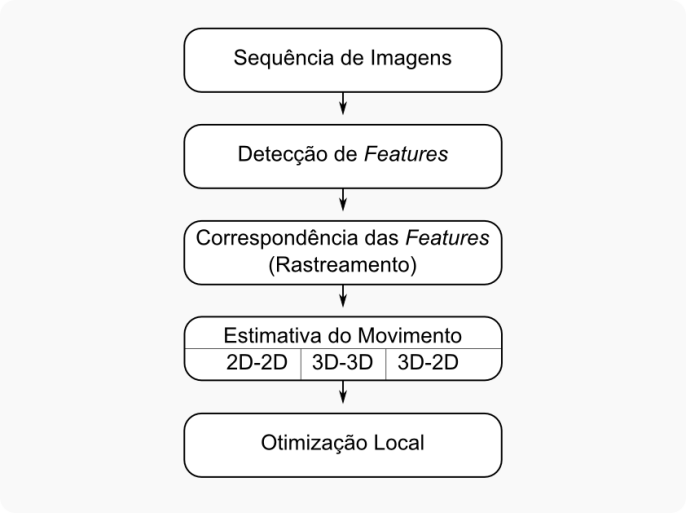


Figura 14 – Fluxo de execução da odometria visual.

significa que houve uma correspondência (Figura 15c).

A estimativa do movimento é a parte central da *VO*. Scaramuzza e Fraundorfer (2011) citam que há dois tipos principais de abordagem para computar o movimento relativo (T_k): métodos baseados em aparência e métodos baseados em *features*. Os métodos baseados em aparência utilizam a intensidade da informação de todos os *pixels* nas duas imagens de entrada, são menos precisos e têm um elevado custo computacional. Os métodos baseados em *features* utilizam somente *features* que têm destaque e que facilmente se repetem nas detecções ao longo das imagens capturadas e é necessário o uso de um algoritmo de *matching* (correspondência) robusto. Métodos baseados em *features*, mesmo com um alto custo computacional do *matching*, são mais rápidos e precisos que os métodos baseados em aparência.

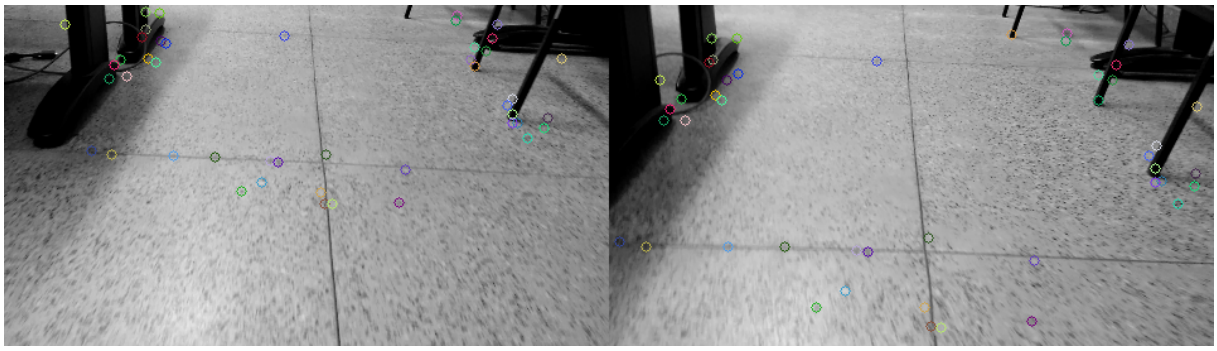
Os métodos baseados em *features* por sua vez, podem ser:

- **2D-2D:** ambas as *features* da correspondência estão especificadas como coordenadas 2D na imagem;
- **3D-3D:** ambas as *features* da correspondência estão especificadas como coordenadas 3D, sendo necessário, a cada instante, realizar uma triangulação 3D dos pontos (por exemplo, sistema de câmeras com visão estéreo);
- **3D-2D:** nesse caso, as *features* da primeira imagem estão especificadas como coordenadas 3D e suas correspondências são representadas por reprojeções 2D na segunda imagem (um exemplo são sistemas de realidade aumentada).

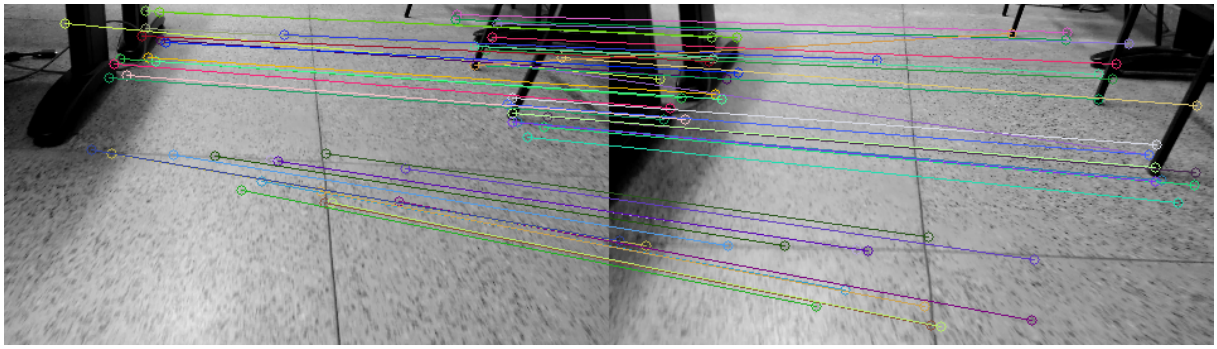
Um maior detalhamento sobre o desenvolvimento matemático de cada método pode ser encontrado em Scaramuzza e Fraundorfer (2011).



(a)



(b)



(c)

Figura 15 – Imagens ao lado esquerdo foram capturadas no tempo k e ao lado direito no tempo $k+1$. a) Sequência de imagens capturadas pela câmera. b) Detecção e descrição das *features*. c)Correspondência das *features* detectadas.

A última etapa consiste na otimização local. Como a *VO* funciona por meio do cálculo da trajetória feita pela câmera de maneira incremental, esta fica sujeita ao acúmulo de erro a cada pose. Esse erro representa um deslize da pose estimada da câmera com relação a sua posição real e pode ser minimizado através da aplicação de uma técnica chamada de ajuste de grupo de janelas deslizantes ou por meio da fusão de um outro sensor como *GPS*, sensor de distância a laser e até mesmo um sensor de medidas inercial (*Inertial Measurement Unit*, *IMU*), que fornece dados de acelerômetro, giroscópio e bússola ao robô.

2.3 Localização *indoor* com *WiFi*

Com a popularização das redes sem fio *WiFi*, muitos trabalhos na área de localização têm surgido utilizando leituras da força do sinal *WiFi* emitido pelos pontos de acesso. *WiFi* é uma marca registrada da *Wi-Fi Alliance* e é utilizada para certificar produtos que pertencem à classe de dispositivos de rede local sem fios (*WLAN*) baseados no padrão *IEEE* (*Institute of Electrical and Electronic Engineers*) 802.11.

Muitas abordagens têm sido criadas no processo de localização baseadas na propagação do sinal das redes sem fio. Liu et al. (2007) cita que os métodos de localização utilizando sinais *wireless indoor* podem ser classificados baseados em duas características: quanto ao algoritmo para localizar a posição (o método que determina a localização, utilizando diversos tipos de medições como *Time Of Flight*, ângulo e força do sinal) e quanto à parte física ou infraestrutura do sensor utilizado (a tecnologia *wireless* envolvida na comunicação com os dispositivos móveis ou estáticos). Liu et al. (2007) diz ainda que os métodos de localizar a posição a partir do sinal *wireless* são divididos da seguinte maneira:

- **Trilateração**: métodos que utilizam as propriedades geométricas do triângulo para encontrar a posição do alvo. Podem ser divididos em métodos por lateração e angulação;

  ◦ **Lateração**: estima a posição do objeto através da leitura de distâncias a partir de múltiplos pontos de referência:

  -*TOA* (*Time Of Arrival*): a distância do alvo móvel até a unidade medidora é diretamente proporcional ao tempo de propagação;

  -*TDOA* (*Time Difference Of Arrival*): a ideia básica desse método é determinar a posição relativa do transmissor através da diferença de tempo que o sinal chega nas unidades receptoras;

  -Baseadas em *RSS* (*Received Signal Strength*): como os métodos *TOA* e *TDOA* necessitam da inexistência de obstáculos entre o emissor e o receptor, os

métodos baseados em *RSS* tentam calcular a posição baseada na atenuação do sinal durante a propagação;

***-RTOF* (*Reflection Time Of Flight*)**: esse método utiliza o tempo de voo do sinal, medido a partir do emissor até a chegada ao receptor, para estimar a posição;

***-RSP* (*Received Signal Phase*)**: utiliza a diferença de fase no sinal das portadoras (de telefonia, por exemplo) para estimar a posição.

- **Angulação**: estima a posição do alvo através da interseção de vários pares de ângulos das linhas de direção. Cada linha é formada pelo raio do círculo de alcance do sinal partindo da estação base ao receptor móvel.

- **Análise de Cenário**: esse termo se refere aos métodos que primeiro coletam características do sinal *wireless* no ambiente e depois estimam a localização de um objeto através da correlação da informação do sinal atual com o sinal adquirido anteriormente. A utilização de um método baseado em *RSS* é comum na etapa de análise do cenário.

Biswas e Veloso (2010) comentam que devido à complexidade de iterações do sinal de *WiFi* em ambientes fechados, métodos baseados em mapas, orientados por dados de intensidade do sinal (Figura 16), têm sido cada vez mais convenientes. O processo de estimar a pose com sinal *WiFi* se baseia na intensidade da força do sinal (*RSS*) que chega ao receptor. Biswas e Veloso (2010) propõem um método probabilístico onde a pose é estimada em um processo de duas etapas: fase de aprendizado e fase de localização. A fase de aprendizado é definida como a construção de um mapa na forma de grafo, onde cada vértice armazena um conjunto de médias e desvios padrão das forças do sinal de cada ponto de acesso (*Access Points*, *AP*) *WiFi* mensurada naquela posição e cada aresta do mapa representa um caminho possível para movimentação do robô.

Em seu trabalho para a etapa de localização, Biswas e Veloso (2010) utilizam o algoritmo que se baseia em filtro de partículas para a obtenção da pose do robô, relacionando as medições do sinal *WiFi* recebido durante a movimentação com os valores previamente populados no mapa. Na Figura 16 é apresentado um exemplo de mapa orientado por intensidade de sinal *WiFi*, onde cada ponto representa um conjunto de médias e desvios padrão das forças do sinal de cada ponto de acesso adquirido na fase de aprendizado.

A Figura 16 demonstra um mapa de intensidade de sinal *RSS*. Cada vértice do mapa corresponde à intensidade do sinal em *dBm* (decibel miliwatt) medido na posição cartesiana x e y durante a etapa de treinamento.

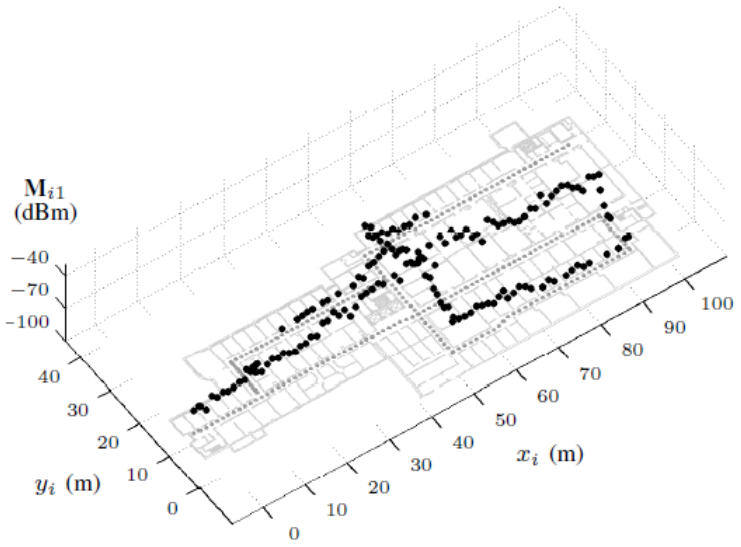


Figura 16 – Exemplo do mapa orientado por intensidade de sinal *WiFi* (BISWAS; VELOSO, 2010).

2.4 Filtro de Bayes

O algoritmo mais geral para o cálculo de *belief* (confiança) é dado pelo algoritmo do filtro de Bayes. Filtros Bayesianos são usados para controlar a distribuição de probabilidade de uma variável $x \in \mathbb{R}^d$ de estado d-dimensional ao longo do tempo, dado um conjunto de observações $z_{1:t}$ e entradas de controle $u_{1:t}$ passadas. A distribuição de probabilidade resultante de x é chamado de *belief* (confiança ou crença), $bel(x_t) = p(x_t|z1:t, u_{1:t})$. A confiança reflete o conhecimento interno do robô sobre o estado do ambiente. O sistema de equações 2.6 (THRUN; BURGARD; FOX, 2005) é utilizado para atualizar recursivamente o valor da *belief*,

$$\begin{cases} \overline{bel}(x_t) = \int p(x_t|u_t, x_{t-1}) bel(x_{t-1}) dx \\ bel(x_t) = \eta p(z_t|x_t) \overline{bel}(x_t) \end{cases} \tag{2.6}$$

onde η é o valor de normalização do filtro. A distribuição de probabilidade $\overline{bel}(x_t)$ é constantemente referida como etapa de predição no contexto de filtro probalístico. Ela prediz o estado x no tempo t baseado no estado anterior $t-1$ antes de incorporar as medições z no tempo t. Calcular o $bel(x_t)$ a partir do $\overline{bel}(x_t)$ é a chamada etapa de atualização ou correção através de medições.

2.5 Filtro de Partículas

O filtro de Partículas (DOUCET et al., 2000) é uma implementação não-paramétrica alternativa do Filtro de Bayes, aproximando o posterior por um número finito de parâmetros.

A posteriori, na estatística *Baysiana*, é a probabilidade condicional que é atribuída após uma medição ser realizada. A ideia chave do filtro de partículas é representar a posteriori $bel(x_t)$ por meio de um conjunto randômico de amostras com estados obtidos a partir deste posteriori. Assim, ao invés de representar a distribuição de forma paramétrica, o filtro de partículas representa a distribuição por um conjunto de amostras tiradas dessa distribuição.

Esse tipo de representação é aproximada, mas é não-paramétrica e, portanto, pode representar um espaço muito mais amplo que as gaussianas, por exemplo. O Algoritmo 1 apresenta uma das mais utilizadas variantes do Filtro de Partículas baseado na amostragem por importância.

Algoritmo 1: Filtro de Partículas

Entrada: X_{t-1}, u_t, z_t

Saída: X_t

1: $\bar{X}_t = X_t = \emptyset$
2: **para** $m = 1 \rightarrow M$ **faça**
3: amostragem $x_t^{[m]} \sim p(x_t \mid u_t, x_{t-1}^{[m]})$
4: $w_t^{[m]} = p(z_t \mid x_t^{[m]})$
5: $\bar{X}_t = \bar{X}_t + \langle x_t^{[m]}, w_t^{[m]} \rangle$
6: **fim para**
7: **para** $m = 1 \rightarrow M$ **faça**
8: obter i com probabilidade $\propto w_t^{[i]}$
9: adicionar $x_t^{[i]}$ em X_t
10: **fim para**

No filtro de partículas, as amostras da distribuição da confiança no posteriori são chamadas de partículas e são indicadas por $X_t = \{x_t^1, x_t^2, x_t^3\}, \ldots, x_t^M$, sendo ^{M} o número total de amostras. Cada partícula x_t^m representa uma hipótese de como seria o estado real do sistema no momento t. Dado que as partículas representam a distribuição da crença a posteriori, teremos uma região com alta concentração de partículas, próxima ao valor esperado dessa distribuição e, ao nos afastarmos desse valor, essa concentração irá diminuir.

O algoritmo do filtro de partículas recebe como entrada um conjunto de partículas que representa a crença sobre o estado anterior $x_{(t-1)}$, o controle u_t e as medições sensoriais z_t no momento t. Então, cria-se um novo conjunto de partículas $\bar{X}_t$ a partir do conjunto anterior $\bar{X}_{t-1}$ aplicando o modelo que simula a ação u_t e adiciona-se um ruído aleatório que representa a incerteza da ação. A seguir, faz-se a medição sensorial e atribui-se pesos às partículas x_t de acordo com a probabilidade da medição sensorial z_t. Faz-se então a reamostragem das partículas com base em seus pesos. Assim, depois de algumas iterações teremos muitas partículas com baixo peso e poucas partículas com alto peso (próximas das

zonas com alta probabilidade). Por fim, as partículas com baixo peso (afastadas das zonas de alta probabilidade) serão descartadas e substituídas por novas partículas, colocadas nas zonas de alta probabilidade. A Figura 17 ilustra as etapas do filtro de partículas.

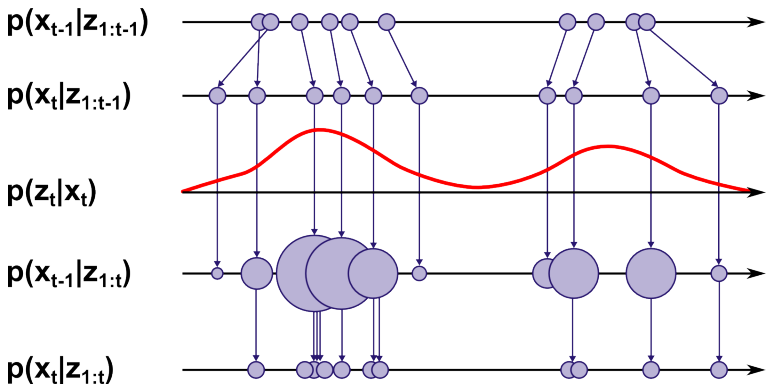


Figura 17 – Etapas do filtro de partículas. Adaptado de (THRUN; BURGARD; FOX, 2005)

Uma das grandes vantagens do filtro de partículas deve-se à sua capacidade de trabalhar com distribuições multimodais. Contudo, este método possui uma elevada complexidade computacional.

2.6 Android

Com o passar dos tempos os telefones celulares foram evoluindo, ganhando cada vez mais recursos e se tornando um item quase indispensável na vida das pessoas. Também houve uma mudança significativa para os desenvolvedores de software, pois o mercado de desenvolvimento para celulares era praticamente restrito aos fabricantes e operadoras que controlavam a criação e inclusão dos aplicativos em seus aparelhos. Dentre os vários sistemas operacionais (SOs) disponíveis atualmente, o Android se destaca (GOOGLE, 2015).

O SO Android é baseado em Linux e primariamente destinado a dispositivos *touchscreen* como smartphones e tablets. Mantido pela empresa Google, o Android tem um papel de destaque no mercado tanto pela quantidade significativa de dispositivos produzidos como por oferecer uma *API* (*Application Programming Interface*) rica, disponibilizando fácil acesso a vários recursos de hardware, tais como câmera, *WiFi*, acelerômetro, bússola e *GPS*, além de boas ferramentas para o desenvolvedor. A facilidade de desenvolver utilizando linguagem de programação Java bastante disseminada e a quantidade de dispositivos Android em uso faz com que a plataforma seja bastante popular.

O SO Android permite também o desenvolvimento de aplicações nativa para Android. Estas aplicações utilizam código Java como base e possuem acessos para operações nativas através de código C++, que é compilado para uma biblioteca e chamado através do

código Java utilizando o método *Java Native Interface* (*JNI*). Por ser um sistema aberto e bem difundido, o Android possui uma grande quantidade de bibliotecas desenvolvidas e bem documentadas que facilitam o desenvolvimento de aplicações para esse sistema. Uma biblioteca de visão computacional bastante conhecida para Android é a *OpenCV*.

2.7 *OpenCV*

OpenCV (*Open source Computer Vision library*) é uma biblioteca de código aberto com mais de 500 algoritmos otimizados para processamento e análise de imagem e vídeo. Desde a sua introdução tem sido amplamente adotada como ferramenta de desenvolvimento pela comunidade de pesquisadores e desenvolvedores em visão computacional.

O *OpenCV* (ITSEEZ, 2015) foi originalmente desenvolvido na Intel, como uma iniciativa para avançar a pesquisa em visão e promover o desenvolvimento de aplicativos com uso intensivo da *CPU* (*Central Processing Unit*) baseados em visão. Após uma série de versões beta, a versão 1.0 foi lançada em 2006. Um segundo grande momento ocorreu em 2009 com o lançamento do *OpenCV* 2 que propôs importantes mudanças, especialmente a nova interface em C++ e está atualmente na verão 3.1 que inclui algoritmos e otimizações como *Structure from Motion*, não presente nas versões anteriores.

Um dos objetivos da *OpenCV* é fornecer uma infra-estrutura de visão para computador que seja simples de usar e que ajude as pessoas a construírem aplicações de visão sofisticadas rapidamente. A biblioteca *OpenCV* contém funções que abrangem muitas áreas, incluindo inspeção de produtos em fábrica, processamento de imagens médicas, segurança, interface de usuário, calibração da câmera, visão estéreo e robótica. Devido a visão computacional e o aprendizado de máquina muitas vezes atuarem em conjunto, o *OpenCV* também contém uma biblioteca de propósito geral, de Aprendizado de Máquina (*Machine Learning*, *ML*). Esta sub-biblioteca é focada em reconhecimento de padrões estatísticos e *clustering*. A biblioteca de *ML* é muito útil para as tarefas de visão que estão no núcleo da *OpenCV*, mas é geral o suficiente para ser usada em qualquer problema envolvendo aprendizagem de máquina.

Na robótica, a biblioteca *OpenCV* atua como uma ferramenta de visão computacional. Algoritmos de detecção, descrição e classificação de características contribuem para que o robô consiga perceber o ambiente através de imagens obtidas por uma câmera.

2.8 Considerações Finais

Neste capítulo foi abordada a base teórica encontrada na literatura para os métodos utilizados neste trabalho, como métodos de estimativa de deslocamento, métodos de

sensoriamento do ambiente e filtros probabilísticos. No Capítulo 3 será explanada a proposta utilizada para o sistema de localização que será desenvolvido neste trabalho.

3 Sistema Proposto

Este trabalho apresenta um sistema de localização *indoor* utilizando filtro probabilístico integrando odometria visual e localização baseada em sinal *WiFi*. O robô utilizado nos experimentos é composto de um smartphone Android, uma plataforma com dois motores de acionamento diferencial contendo sensores de *encoder* em cada roda e um *hardware* embarcado. O smartphone será responsável pela execução das etapas de sensoriamento, processamento odométrico visual e da localização e o *hardware* embarcado será responsável pela parte de acionamento dos motores.

O sistema segue o fluxo presente na Figura 18. Inicialmente o robô realiza um deslocamento no ambiente, ao longo do deslocamento quadros de imagem são capturados sequencialmente obtidos pela câmera do smartphone, os quadros são processados pelo algoritmo da odometria visual e seu deslocamento é computado. O robô então obtêm a orientação da bússola no sensor *IMU* do smartphone e faz uma leitura de todos os *APs* em seu alcance. Em seguida as leituras obtidas nos sensores são aplicadas às partículas do filtro probabilístico e sua nova pose é estimada. A nova pose corresponderá à localização do robô no ambiente.

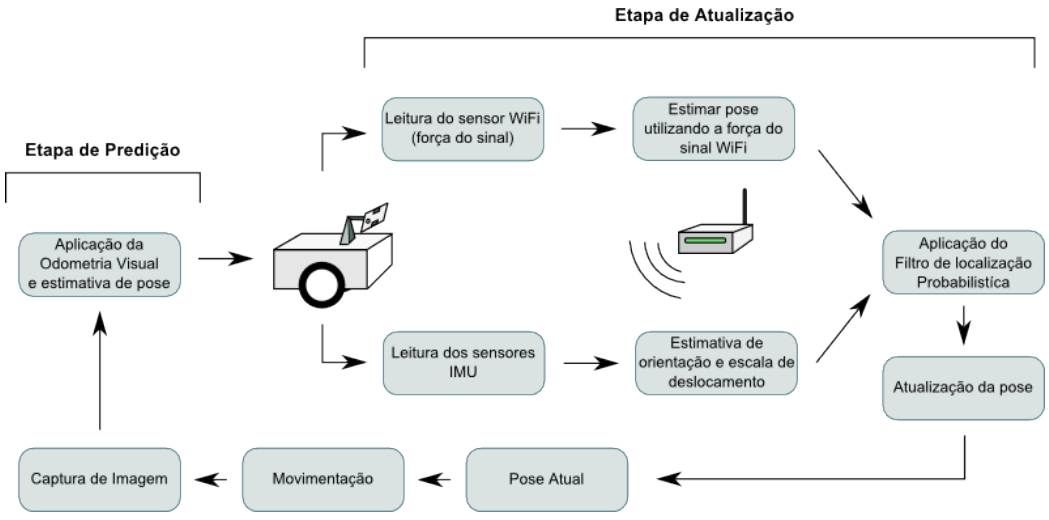


Figura 18 – Fluxo de execução do sistema de localização.

Para este trabalho foi selecionado o método de odometria visual citado no capítulo 2. O procedimento adotado para a *VO* segue a formulação proposta por Scaramuzza e Fraundorfer (2011), utilizando um sistema monocular e método baseado em *features*. As imagens utilizadas na odometria visual serão adquiridas da câmera do smartphone e submetidas a um processo de detecção, descrição e correspondência. Uma vez encontradas as *features*, será utilizada geometria epipolar que, de acordo com (HARTLEY; ZISSERMAN, 2004), é a projeção geométrica intrínseca entre duas visões, para extrair a Matriz de

Transformação T, (Equação 3.1), entre duas imagens, usadas para estimar a posição da câmera. Onde R é a Matriz de Rotação e t é a Matriz de Translação da perspectiva das imagens *k-1* a *k*.

$$T_{k,k-1} = \begin{bmatrix} R_{k,k-1} & t_{k,k-1} \\ 0 & 1 \end{bmatrix} \tag{3.1}$$

Uma vez obtida a matriz de rotação e o vetor de translação realizada pela câmera, é possível estimar o deslocamento realizado entre o momento que o primeiro e o segundo quadro de imagem foram capturados pelo robô.

Foi feita uma análise da eficiência e viabilidade da utilização dos seguintes algoritmos (presentes na biblioteca *OpenCV*), dado seu custo computacional, pois o *hardware* utilizado tem suas limitações. A análise foi dividida em duas partes:

- Análise 1 (Utilizando *OpenCV* 2.4.9):
  - Algoritmos de detecção de *features*: *FAST* (ROSTEN; REITMAYR; DRUMMOND, 2005), *STAR* (AGRAWAL; KONOLIGE, 2008), *SIFT* (LOWE, 1999), *SURF* (BAY et al., 2008), *ORB* (RUBLEE et al., 2011), *BRISK* (LEUTENEGGER; CHLI; SIEGWART, 2011), *MSER* (MATAS et al., 2002), *GFTT* (SHI; TOMASI, 1993), *HARRIS* (HARRIS; STEPHENS, 1988), *Dense* (OPENCV, b), *SimpleBlob* (OPENCV, c);
  - Algoritmos de descrição: *FREAK* (ALAHI; ORTIZ; VANDERGHEYNST, 2012), *BRISK*, *BRIEF* (CALONDER et al., 2010), *SIFT*, *SURF*, *ORB*;
  - Algoritmos de correspondência: *BruteForce-L2*, *BruteForce-L1*, *BruteForce-Hamming* (OPENCV, a).
- Análise 2 (Utilizando *OpenCV* 3.0.0):
  - Algoritmos de detecção de *features*: *AGAST* (MAIR et al., 2010), *AKAZE* (ALCANTARILLA; NUEVO; BARTOLI, 2013), *BRISK*, *FAST*, *GFTT (Good Features to Track)*, *HARRIS*, *KAZE* (ALCANTARILLA; BARTOLI; DAVISON, 2012), *MSER* e *ORB*;
  - Algoritmos de descrição: *AKAZE*, *BRISK* e *ORB*;
  - Algoritmos de correspondência: *BruteForce* (*BF*), *BruteForce-Hamming* (*BFH*) e *BruteForce-Hamming Look-up Table* (*LUT*).

A análise ocorreu em duas etapas devido a mudança da versão da *OpenCV*, que forneceu otimizações nos algoritmos já existentes, acrescentou e removeu algoritmos proprietários. Alguns algoritmos, como o *SIFT* e o *SURF*, não foram selecionados para a

segunda etapa da análise devido ao seu tempo de execução muito elevado. Mais detalhes sobre os tempos de execução se encontram no capítulo 4. As análises 1 e 2 foram executadas respectivamente nos dispositivos especificados nas Tabelas 1 e 2.

Tabela 1 – Principais características dos smartphones utilizados no projeto durante a Análise 1.

Modelo	**Processador**	**RAM**	**Versão do SO**
Moto XT1058	Dual-Core 1.7 GHz	2 GB	Android 4.4.4 (KitKat)
LG Nexus 5	Quad-Core 2.3 GHz	2 GB	Android 4.4.4 (KitKat)
Moto XT1097	Quad-Core 2.5 GHz	2 GB	Android 5.1 (Lollipop)

Tabela 2 – Principais características dos smartphones utilizados no projeto durante a Análise 2.

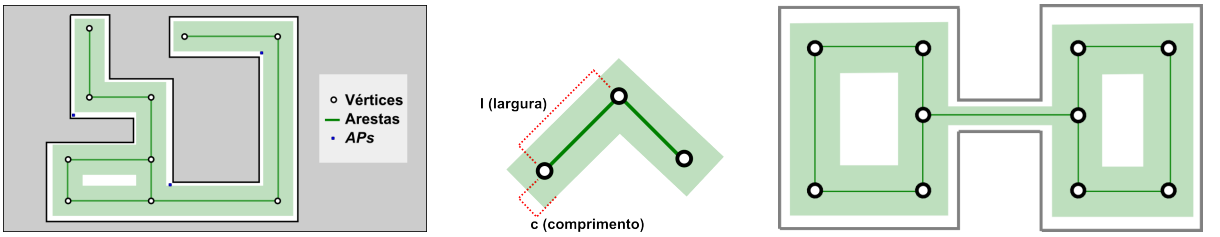
Modelo	**Processador**	**RAM**	**Versão do SO**
Samsung SM-T520	Quad-Core 1.9GHz	2 GB	Android 4.4 (KitKat)
LG Nexus 5	Quad-Core 2.3GHz	2 GB	Android 6.0.1 (Marshmallow)
Moto XT1097	Quad-Core 2.5GHz	2 GB	Android 5.1 (Lolipop)

Segundo a documentação da biblioteca *OpenCV*, os algoritmos *BruteForce-L1* e *BruteForce-L2* melhor se adequam aos descritores *SIFT* e *SURF*, pois são descritores que utilizam estruturas de ponto flutuante. Os métodos *BruteForce*, *BruteForce-Hamming* e *BruteForce-Hamming Look-up Table* se aplicam aos descritores *FREAK*, *BRISK*, *BRIEF* e *ORB*, pois estes trabalham com descrições baseadas em *strings* binárias. Os algoritmos citados foram escolhidos por serem algoritmos clássicos na literatura utilizados para extração e *matching* de *features*. O descritor *AKAZE*, em especial, somente é compatível com o detector *AKAZE*.

Com objetivo de avaliar a eficiência da odometria visual foi necessário um conjunto de dados que representasse as diversidades de um trajeto como no mundo real. As pesquisas levaram ao conjunto de dados KITTI (GEIGER et al., 2013), que fornece um conjunto de imagens originais e sem distorção de áreas rurais e urbanas, a pose real para comparação e validação e a especificação de calibração e configuração das câmeras utilizadas. As imagens do conjunto de dados têm uma resolução de [1241 x 376] *pixels*, tamanho suficiente para os objetivos do experimento. A ferramenta de simulação V-rep também foi utilizada como fonte de imagens. Através da *API* remota é possível obter imagens sem distorção do ambiente simulado na resolução de [512 x 512] *pixels*.

Neste trabalho foi utilizado um mapa topológico contendo as forças de sinal *WiFi*, proposta por Biswas (BISWAS; VELOSO, 2010) em coordenadas conhecidas, onde o robô deverá estimar a posição no mapa. O mapa é do tipo topológico modelado por um grafo definido por $G = \{V, A\}$, onde G representa o grafo do mapa (Figura 19a); V representa um conjunto de vértices e A representa um conjunto de arestas. A estrutura do vértice V por

sua vez é composto de $V = \{P, M, D\}$, uma tupla que contendo a posição P (coordenada x e y representando a posição do vértice no mundo real), um conjunto M contendo as médias do sinal RSS *WiFi* naquele vértice e um conjunto D contendo o desvio padrão das médias mensuradas de cada AP no vértice. Cada aresta segue a estrutura $A = \{v1, v2, l, c\}$, onde $v1$ e $v2$ correspondem aos vértices que compõem a aresta, l corresponde à distância (largura) entre os vértices e c o comprimento da passagem formada pela aresta (Figura 19b).



(a) Mapa topológico. (b) Representação de uma passagem. (c) Passagens com comprimentos diferentes.

Figura 19 – Exemplo do mapa topológico.

Para a etapa de localização, utilizou-se o filtro de partículas proposto por Thrun, Burgard e Fox (2005). O sensoriamento será feito a partir das leituras *RSS* do *WiFi*, leituras de orientação (bússola) e estimativa de escala de deslocamento (acelerômetro) do *IMU*.

3.1 Simulador V-rep

A simulação computacional permite que diversos testes sejam realizados sem a necessidade de implantação do sistema em robô real garantindo, assim, uma maior economia de recursos (inclusive financeiros). Visto que na simulação não há desgaste de componentes do robô ou outro possível objeto físico utilizado em um teste real, torna-se desnecessário a troca dos mesmos. Na simulação há uma economia de tempo, já que ela pode ser acelerada e também não há necessidade de se perder tempo recarregando as fontes de energia do robô.

A simulação garante uma maior segurança à integridade física do robô, dos objetos do local em que ele está inserido e até mesmo de seres humanos ou outros seres vivos, pois tudo acontece no computador, não há perigo de o robô se chocar com uma obstáculo e danificar sua estrutura ou causar ferimentos a alguém em caso de erro. Ela permite o aperfeiçoamento dos algoritmos testados na simulação, visto que podemos verificar se os resultados são satisfatórios no simulador, antes de aplicar em um robô real.

A aplicação V-Rep (ROHMER S. P. N. SINGH, "2013") (Figura 20) é uma ferramenta que permite que o usuário monte cenários com diversos objetos simuláveis como

robôs, obstáculos e sensores. Ele oferece vários modelos de robôs existentes no mundo real disponíveis para uso bem como ferramentas para editá-los e sua funcionalidade ou para criar novos. O simulador permite a utilização de quatro bibliotecas conhecidas de simulação das físicas do ambiente: *ODE* (*Open Dynamics Engine*) ("OPEN..., "2016"), *Bullet* ("BULLET", "2016"), *Vortex* ("VORTEX", "2016") e *Newton* ("NEWTON..., "2016").

O V-rep foi escolhido por se destacar dos demais simuladores devido a: possuir uma interface bastante amigável, ter uma versão gratuita para estudantes, ter um bom número de sensores e atuadores padrões na robótica, permitir ao usuário escolher com qual biblioteca de físicas deseja realizar a simulação, possuir geração de gráficos em tempo de execução, ser multiplataforma (a partir de versões mais atuais), possuir boa documentação e com exemplos ricos, incluir uma *API* de comunicação remota.

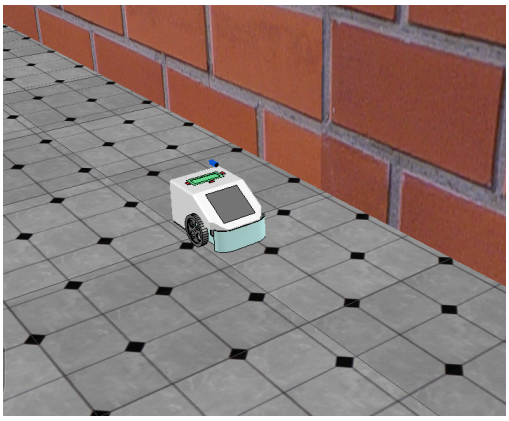


Figura 20 – Robô sendo simulado na ferramenta V-rep.

A análise do sistema proposto foi dividida em três cenários, modelados no simulador V-rep versão 3.3.1:

- Variação da quantidade de *features* utilizadas na odometria visual em um mapa de uma sala;
- Variação da quantidade de partículas do filtro no mapa do DC/UFPI (Departamento de Computação);
- Análise em um robô com tamanho de uma cadeira de rodas e variação da quantidade *APs* em um mapa de uma ala hospitalar baseado no HU/UFPI (Hospital Universitário).

O robô utilizado no simulador foi o modelo DR12 da Cubictek, que possui 2 rodas de acionamento diferencial e dimensões de 0,12 x 0,25 x 0,14 metros (largura x comprimento x altura).

Cada cenário foi analisado nos 3 modelos diferentes de smartphone Android disponíveis para este projeto. A Tabela 2 contém o modelo e as especificações de cada modelo.

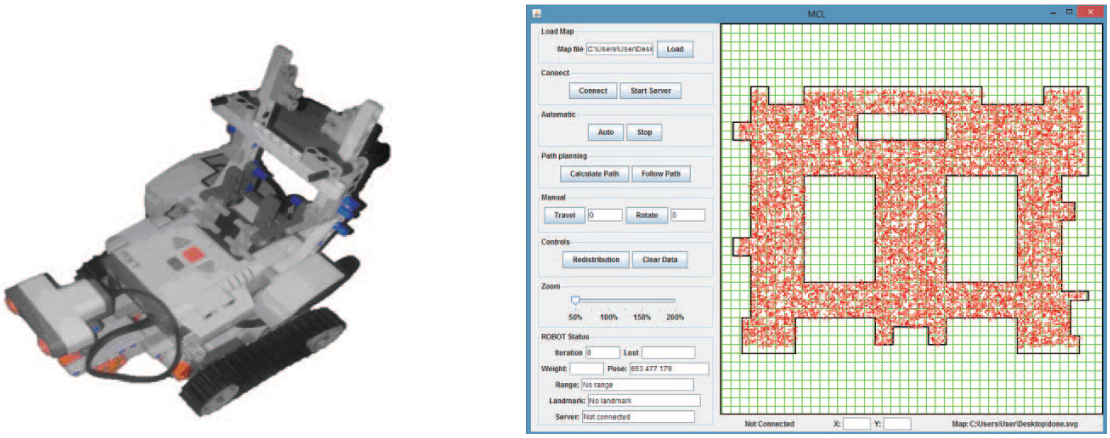
As implementações foram feitas em linguagem Java e C++ utilizando interface *JNI*. A escolha dessas linguagens foi motivada devido ao *SDK* para programação em sistema *Android* fornecer uma *API* em Java robusta e amigável para comunicação com os sensores do smartphone. Existe ainda o *NDK*, que é a *API* nativa em C++ do *Android* que foi necessária para execução de tarefas de uso intensivo de processador com mais rapidez. Neste trabalho foram utilizadas as versões 25.2.0 do *SDK* e r12b do *NDK*. A IDE escolhida para programação foi o Eclipse versão 4.4.2 Luna.

3.2 Trabalhos de Influência

Alguns projetos serviram de inspiração para a execução deste trabalho, dentre eles dois se destacam como principais trabalhos que motivaram no desenvolvimento do sistema de localização.

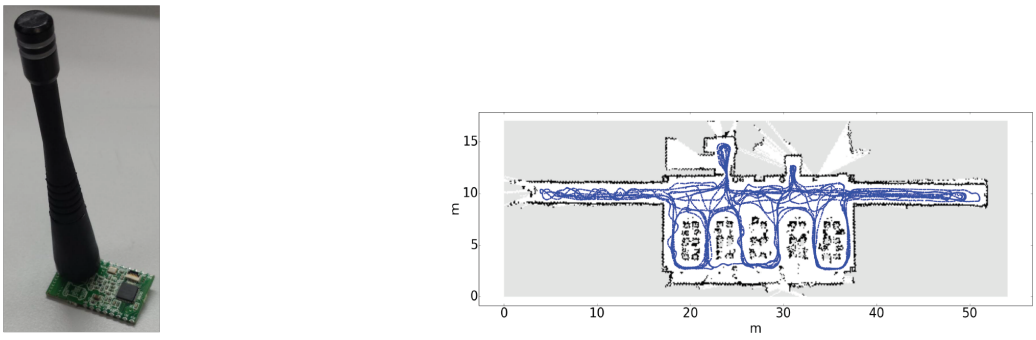
O primeiro trabalho se trata de um sistema de localização *indoor* utilizando algoritmo de Monte Carlo (baseado em filtro de partículas) e reconhecimento de *features*. Lye et al. (LYE et al., 2014) propõem um sistema de navegação para robôs móveis que faz uso de um smartphone Android para reconhecimento de marcos predefinidos através do processamento de imagens, adquiridas pela câmera, com o algoritmo *ORB* da biblioteca *OpenCV*. Este sistema opera em uma plataforma robótica Lego *Mindstorms NXT* (Figura 21a), que é acionada por comandos transmitidos de forma *wireless* a partir de um computador. Os dados de reconhecimento das *features* são enviados do smartphone ao computador por meio de comunicação *bluetooth* e no computador é feito o processamento do algoritmo de localização (Figura 21b). O sistema conta também com um planejador do caminhos baseado no Algoritmo A*.

Canedo-Rodriguez et al. (2015) propuseram em seu trabalho uma análise de um sistema de localização para robôs móveis comparando *motes* de baixo custo e *WiFi*, com localização por filtro de partículas. *Motes* são nós sensores em uma rede de sensores capazes de processar informações, obter dados de algum sensor no nó e se comunicar com os outros nós da rede (Figura 22a). Este trabalho faz uso de odometria mecânica, sensor *LIDAR* (da sigla inglesa *Light Detection And Ranging*) e da estratégia de geração de um mapa probabilístico contendo as médias e desvios do *RSS* de cada *mote* e ponto de acesso *WiFi*. Este trabalho verificou também a influência da variação na força de transmissão de cada mote no processo de localização. Como plataforma robótica, um robô modelo Pionner P3DX, 6 APs WiFi e 6 motes *wireless*.



(a) (b)

Figura 21 – a) Robô construído de Lego; b) Interface do programa de localização no computador.



(a) (b)

Figura 22 – a) Nó sensor, mote; b) Interface do programa de localização no computador.

3.3 Considerações Finais

Neste capítulo foi apresentada a proposta que descreve o funcionamento do sistema de localização criado neste trabalho. O sistema desenvolvido objetiva a execução da localização com um baixo custo computacional, utilizando a fusão sensorial de odometria visual, localização com *RSS WiFi*, orientação através de sensor *IMU* e aplicação de um filtro probabilístico de partículas para realizar a correção da pose do robô. Também foram apresentados os principais trabalhos que influenciaram no desenvolvimento deste projeto. No capítulo 4 serão apresentados os resultados obtidos a partir das análises feitas no sistema de localização.

4 Resultados e Discussão

Neste capítulo são apresentados os resultados obtidos durante a análise do sistema desenvolvido com relação ao custo computacional de execução dos algoritmos para odometria visual e filtro de partículas.

4.1 Análise 1

Durante a Análise 1, utilizando dados reais, foram feitas 100 execuções de cada combinação dos algoritmos (detecção, descrição, correspondência) utilizando os parâmetros padrões em cada dispositivo móvel descrito na proposta, com resultados obtidos na unidade dos segundos. Os resultados da Análise 1 foram dispostos em 9 tabelas, onde na primeira coluna encontram-se os algoritmos de detecção e na primeira linha os algoritmos de descrição, agrupados pelos algoritmos de correspondência, pois este tinha suas restrições quanto aos descritores que eram compatíveis. Os descritores *SIFT* e *SURF*, devido a utilizarem estruturas de ponto flutuante, somente são compatíveis com os algoritmos de correspondência *BruteForce-L1* e *Bruteforce-L2*. O melhor resultado é o tempo mais baixo em cada tabela, este foi destacados em negrito.

Os resultados são apresentados da seguinte forma: As tabelas 3, 4 e 5 correspondem respectivamente aos algoritmos de correspondência *BruteForce-L1*, *BruteForce-L2*, *BruteForce-Hamming* e demonstram as médias e desvios no tempo de execução no smartphone de modelo Motorola XT1058.

Tabela 3 – Tabela com as médias e desvios padrão dos tempos de execução dos algoritmos no Motorola XT1058, utilizando a correspondência por *BruteForce-L1*.

	SIFT	*SURF*
FAST	3,395 ± 0,060	1,523 ± 0,019
STAR	0,154 ± 0,002	**0,061 ± 0,004**
SIFT	1,698 ± 0,018	1,096 ± 0,030
SURF	10,054 ± 0,061	1,973 ± 0,022
ORB	5,013 ± 0,038	1,291 ± 0,018
BRISK	0,334 ± 0,011	0,072 ± 0,004
MSER	0,584 ± 0,011	0,165 ± 0,005
GFTT	0,777 ± 0,006	1,086 ± 0,015
HARRIS	0,780 ± 0,010	1,086 ± 0,019
Dense	2,946 ± 0,023	4,497 ± 0,063
SimpleBlob	0,204 ± 0,008	0,100 ± 0,005

As tabelas 6, 7 e 8 correspondem respectivamente aos algoritmos de correspondência *BruteForce-L1*, *BruteForce-L2*, *BruteForce-Hamming* e demonstram as médias e desvios

Tabela 4 – Tabela com as médias e desvios padrão dos tempos de execução dos algoritmos no Motorola XT1058, utilizando a correspondência por *BruteForce-L2*.

	SIFT	*SURF*
FAST	3,580 ± 0,052	1,632 ± 0,057
STAR	0,155 ± 0,004	**0,062 ± 0,002**
SIFT	1,703 ± 0,023	1,109 ± 0,050
SURF	10,099 ± 0,077	2,007 ± 0,057
ORB	5,033 ± 0,060	1,296 ± 0,018
BRISK	0,332 ± 0,057	0,071 ± 0,002
MSER	0,584 ± 0,008	0,163 ± 0,009
GFTT	0,880 ± 0,011	1,139 ± 0,012
HARRIS	0,884 ± 0,015	1,150 ± 0,058
Dense	4,322 ± 0,050	5,143 ± 0,059
SimpleBlob	0,204 ± 0,012	0,102 ± 0,006

Tabela 5 – Tabela com as médias e desvios padrão dos tempos de execução dos algoritmos no Motorola XT 1058, utilizando a correspondência por *BruteForce-Hamming*

	FREAK	*BRISK*	*BRIEF*	*ORB*
FAST	0,141 ± 0,021	0,175 ± 0,004	0,084 ± 0,008	0,242 ± 0,010
STAR	0,055 ± 0,001	0,055 ± 0,001	0,055 ± 0,001	0,062 ± 0,004
SIFT	0,796 ± 0,013	0,798 ± 0,016	0,793 ± 0,015	-
SURF	0,729 ± 0,014	0,759 ± 0,011	0,748 ± 0,011	0,857 ± 0,010
ORB	**0,028 ± 0,003**	0,041 ± 0,002	0,050 ± 0,005	0,141 ± 0,003
BRISK	0,032 ± 0,002	0,033 ± 0,002	0,033 ± 0,004	0,060 ± 0,002
MSER	0,108 ± 0,001	0,110 ± 0,004	0,108 ± 0,001	0,116 ± 0,001
GFTT	0,154 ± 0,013	0,176 ± 0,015	0,119 ± 0,005	0,229 ± 0,004
HARRIS	0,154 ± 0,004	0,179 ± 0,006	0,121 ± 0,002	0,229 ± 0,007
Dense	0,551 ± 0,009	0,679 ± 0,008	0,262 ± 0,004	0,617 ± 0,009
SimpleBlob	0,091 ± 0,006	0,093 ± 0,007	0,093 ± 0,007	0,099 ± 0,007

no tempo de execução no smartphone de modelo Motorola XT1097.

Tabela 6 – Tabela com as médias e desvios padrão dos tempos de execução dos algoritmos no Motorola XT1097, utilizando a correspondência por *BruteForce-L1*.

	SIFT	*SURF*
FAST	3,214 ± 0,050	2,886 ± 0,017
STAR	0,124 ± 0,005	**0,058 ± 0,002**
SIFT	1,453 ± 0,025	1,327 ± 0,017
SURF	9,031 ± 0,093	3,769 ± 0,023
ORB	3,911 ± 0,050	2,377 ± 0,012
BRISK	0,275 ± 0,006	0,146 ± 0,007
MSER	0,450 ± 0,013	0,217 ± 0,007
GFTT	0,837 ± 0,012	2,022 ± 0,016
HARRIS	0,842 ± 0,012	2,028 ± 0,019
Dense	3,646 ± 0,044	8,752 ± 0,057
SimpleBlob	0,202 ± 0,014	0,147 ± 0,014

Tabela 7 – Tabela com as médias e desvios padrão dos tempos de execução dos algoritmos no Motorola XT1097, utilizando a correspondência por *BruteForce-L2*.

	SIFT	*SURF*
FAST	3,458 ± 0,073	3,073 ± 0,028
STAR	0,124 ± 0,003	**0,059 ± 0,001**
SIFT	1,475 ± 0,027	1,330 ± 0,021
SURF	9,016 ± 0,110	3,848 ± 0,035
ORB	3,935 ± 0,050	2,390 ± 0,018
BRISK	0,278 ± 0,011	0,145 ± 0,007
MSER	0,444 ± 0,015	0,213 ± 0,011
GFTT	0,985 ± 0,011	2,133 ± 0,031
HARRIS	0,986 ± 0,016	2,164 ± 0,032
Dense	4,960 ± 0,023	9,400 ± 0,052
SimpleBlob	0,208 ± 0,016	0,151 ± 0,014

Tabela 8 – Tabela com as médias e desvios padrão dos tempos de execução dos algoritmos no Motorola XT1097, utilizando a correspondência por *BruteForce-Hamming*

	FREAK	*BRISK*	*BRIEF*	*ORB*
FAST	0,163 ± 0,032	0,199 ± 0,016	0,112 ± 0,004	0,282 ± 0,011
STAR	0,045 ± 0,002	0,046 ± 0,003	0,045 ± 0,002	0,050 ± 0,002
SIFT	0,666 ± 0,016	0,673 ± 0,018	0,664 ± 0,021	-
SURF	1,376 ± 0,013	1,418 ± 0,022	1,408 ± 0,023	1,639 ± 0,035
ORB	**0,019 ± 0,003**	0,031 ± 0,004	0,041 ± 0,003	0,141 ± 0,010
BRISK	0,027 ± 0,002	0,026 ± 0,002	0,027 ± 0,001	0,052 ± 0,004
MSER	0,078 ± 0,003	0,078 ± 0,001	0,078 ± 0,003	0,086 ± 0,005
GFTT	0,166 ± 0,004	0,207 ± 0,003	0,115 ± 0,003	0,238 ± 0,009
HARRIS	0,150 ± 0,008	0,200 ± 0,011	0,115 ± 0,004	0,243 ± 0,012
Dense	0,667 ± 0,028	0,827 ± 0,032	0,366 ± 0,026	0,721 ± 0,014
SimpleBlob	0,120 ± 0,011	0,121 ± 0,011	0,123 ± 0,011	0,128 ± 0,011

As tabelas 9, 10 e 11 correspondem respectivamente aos algoritmos de correspondência *BruteForce-L1*, *BruteForce-L2*, *BruteForce-Hamming* e demonstram as médias e desvios no tempo de execução no smartphone de modelo LG Nexus 5.

Esta etapa da execução foi feita utilizando um robô real e em todas as execuções foram utilizadas imagens (figura 23) obtidas pela câmera do smartphone acoplada ao robô, com resolução 640 x 480 pixels, com tempos de captura diferentes durante a movimentação. Para os algoritmos foram utilizados os parâmetros padrões fornecidos pela biblioteca. Nesse teste inicial, a melhor combinação entre algoritmos foi a de *ORB-FREAK-BruteForce-Hamming* (detecção - descrição - correspondência) com média de 0,019 segundos e desvio padrão de 0,003 segundos no smartphone Motorola XT1097. Destacam-se também as combinações *BRISK-BRIEF-BruteForce-Hamming* e *BRISK-BRISK-BruteForce-Hamming* com médias de tempo 0,029 segundos no smartphone LG Nexus 5.

A combinação *SIFT-ORB-BruteForce-Hamming* não foi possível ser testada devido

Tabela 9 – Tabela com as médias e desvios padrão dos tempos de execução dos algoritmos no Nexus 5, utilizando a correspondência por *BruteForce-L1*.

	SIFT	*SURF*
FAST	2,351 ± 0,061	0,642 ± 0,038
STAR	0,131 ± 0,003	**0,049 ± 0,001**
SIFT	1,273 ± 0,022	0,771 ± 0,019
SURF	6,953 ± 0,366	0,879 ± 0,120
ORB	3,520 ± 0,198	0,550 ± 0,036
BRISK	0,276 ± 0,007	0,054 ± 0,006
MSER	0,413 ± 0,008	0,124 ± 0,008
GFTT	0,592 ± 0,036	0,495 ± 0,007
HARRIS	0,637 ± 0,017	0,493 ± 0,005
Dense	1,732 ± 0,212	2,163 ± 0,012
SimpleBlob	0,167 ± 0,008	0,088 ± 0,010

Tabela 10 – Tabela com as médias e desvios padrão dos tempos de execução dos algoritmos no Nexus 5, utilizando a correspondência por *BruteForce-L2*.

	SIFT	*SURF*
FAST	2,541 ± 0,021	0,807 ± 0,086
STAR	0,130 ± 0,000	**0,049 ± 0,001**
SIFT	1,297 ± 0,017	0,778 ± 0,018
SURF	6,880 ± 0,044	1,020 ± 0,102
ORB	3,434 ± 0,024	0,714 ± 0,016
BRISK	0,275 ± 0,010	0,053 ± 0,005
MSER	0,408 ± 0,003	0,116 ± 0,006
GFTT	0,685 ± 0,017	0,641 ± 0,010
HARRIS	0,679 ± 0,014	0,633 ± 0,034
Dense	2,157 ± 0,018	2,475 ± 0,011
SimpleBlob	0,166 ± 0,008	0,088 ± 0,009

Tabela 11 – Tabela com as médias e desvios padrão dos tempos de execução dos algoritmos no Nexus 5, utilizando a correspondência por *BruteForce-Hamming*

	FREAK	*BRISK*	*BRIEF*	*ORB*
FAST	0,085 ± 0,020	0,101 ± 0,006	0,059 ± 0,003	0,209 ± 0,006
STAR	0,045 ± 0,001	0,046 ± 0,001	0,046 ± 0,002	0,051 ± 0,000
SIFT	0,597 ± 0,012	0,604 ± 0,008	0,594 ± 0,008	-
SURF	0,308 ± 0,024	0,355 ± 0,031	0,423 ± 0,010	0,517 ± 0,054
ORB	**0,024 ± 0,001**	0,033 ± 0,003	0,041 ± 0,001	0,153 ± 0,007
BRISK	0,034 ± 0,001	0,029 ± 0,001	0,029 ± 0,002	0,046 ± 0,001
MSER	0,081 ± 0,001	0,082 ± 0,001	0,082 ± 0,001	0,087 ± 0,003
GFTT	0,102 ± 0,006	0,114 ± 0,006	0,083 ± 0,004	0,218 ± 0,019
HARRIS	0,113 ± 0,008	0,125 ± 0,005	0,097 ± 0,005	0,212 ± 0,021
Dense	0,272 ± 0,009	0,399 ± 0,008	0,183 ± 0,008	0,530 ± 0,056
SimpleBlob	0,080 ± 0,007	0,079 ± 0,008	0,080 ± 0,007	0,084 ± 0,003



Figura 23 – Imagens utilizadas na análise dos algorítimos para *VO*.

a um problema de compatibilidade da opencv versão 2.4.9, citado pela própria *OpenCV* em sua implementação. Outro problema durante os testes, revelado através do monitoramento da frequência do dispositivo, foi que o uso intensivo do processador pelos algoritmos, após algumas horas, causou uma elevação da temperatura dos dispositivos e uma redução da frequência do processador. Essa redução ocorre devido à tecnologia *ThermalEngine* do sistema Android com finalidade de preservar o sistema de danos causados por temperaturas elevadas. Essa funcionalidade só pode ser desabilitada em dispositivos com acesso ao *root* do sistema. Para isso uma técnica de resfriamento simples, utilizando uma ventoinha, foi suficiente para solucionar este obstáculo. Os resultados apresentados não estão afetados pela limitação de frequência imposta pela *ThermalEngine*.

4.2 Análise 2

A segunda bateria de análises foi realizada utilizando imagens reais do conjunto de dados KITTI. De todos os 21 conjuntos de imagens presentes na base de dados KITTI, foi escolhido o primeiro (chamado de *sequence* 00) para os testes, pois o mesmo possui pouca variância de luminosidade do ambiente (ver Figura 24). Também foi determinado um número máximo de 300 quadros de imagem em cada teste, que seriam o suficiente para calcular a precisão e desempenho. As imagens da base KITTI são imagens obtidas no mundo real utilizando a plataforma de direção autônoma *AnnieWAY*.

Diferente da Análise 1, onde foram avaliados somente as etapas de detecção, descrição e correspondência, os testes da Análise 2 foram feitos calculando o tempo de execução dos algoritmos desde após a captura da imagem, até o resultado do algoritmo da odometria visual, que é o retorno do deslocamento realizado entre os quadros de imagem. Os algoritmos utilizados foram testados com seus parâmetros padrões e por motivo de compatibilidade o descritor *AKAZE* foi utilizado somente com o detector *AKAZE*. Os resultados da Análise 2 foram dispostos em 9 tabelas, onde na primeira coluna encontram-se os algoritmos de detecção e na primeira linha os algoritmos de descrição e cada tabela representa a combinação com um dos algoritmos de correspondência.

Os resultados a seguir são apresentados da seguinte forma: As tabelas 12, 13 e 14 correspondem respectivamente aos algoritmos de correspondência *BruteForce*, *BruteForce-*



Figura 24 – Imagens da base KITTI utilizadas na análise dos algorítimos para *VO*.

Hamming, *BruteForce-Hamming Look-up Table* e demonstram as médias e desvios no tempo de execução no smartphone de modelo Samsung SM-T520.

Tabela 12 – Tabela com as médias e desvios padrão em milissegundos dos tempos de execução dos algoritmos no Samsung SM-T520, utilizando a correspondência por *BruteForce*

	AKAZE	*BRISK*	*ORB*
AGAST	-	10254,230 ± 1987,890	8898,037 ± 1365,514
AKAZE	3834,555 ± 519,147	2722,897 ± 557,066	2473,548 ± 317,722
BRISK	-	4662,616 ± 594,518	4743,949 ± 441,164
FAST	-	10625,183 ± 1815,642	7360,466 ± 1008,505
GFTT	-	1800,128 ± 68,972	1454,202 ± 267,802
HARRIS	-	1771,899 ± 122,920	1355,365 ± 273,586
KAZE	-	6355,813 ± 670,704	5688,769 ± 523,449
MSER	-	2456,073 ± 204,197	2486,476 ± 238,025
ORB	-	**462,192 ± 132,846**	1307,958 ± 363,390

Tabela 13 – Tabela com as médias e desvios padrão em milissegundos dos tempos de execução dos algoritmos no Samsung SM-T520, utilizando a correspondência por *BruteForc-Hamminge*

	AKAZE	*BRISK*	*ORB*
AGAST	-	7504,944 ± 1164,257	6889,010 ± 1392,876
AKAZE	2950,158 ± 435,473	2047,705 ± 394,405	1904,430 ± 129,630
BRISK	-	3599,680 ± 514,543	4105,164 ± 559,109
FAST	-	6667,689 ± 778,295	4571,192 ± 883,931
GFTT	-	1692,001 ± 141,771	827,116 ± 121,517
HARRIS	-	1635,955 ± 193,886	820,331 ± 125,280
KAZE	-	5526,373 ± 514,169	4870,143 ± 171,410
MSER	-	2393,385 ± 232,662	2184,064 ± 337,812
ORB	-	**392,442 ± 105,122**	681,841 ± 222,810

As tabelas 15, 16 e 17 correspondem respectivamente aos algoritmos de correspondência *BruteForce*, *BruteForce-Hamming*, *BruteForce-Hamming Look-up Table* e demonstram as médias e desvios no tempo de execução no smartphone de modelo LG Nexus 5.

As tabelas 18, 19 e 20 correspondem respectivamente aos algoritmos de correspondência *BruteForce*, *BruteForce-Hamming*, *BruteForce-Hamming Look-up Table* e de-

Tabela 14 – Tabela com as médias e desvios padrão em milissegundos dos tempos de execução dos algoritmos no Samsung SM-T520, utilizando a correspondência por *BruteForce-Hamming Look-up Table*

	AKAZE	*BRISK*	*ORB*
AGAST	-	7950,248 ± 1245,786	7157,864 ± 1434,791
AKAZE	2951,303 ± 429,194	2009,210 ± 369,712	1908,377 ± 122,795
BRISK	-	3600,211 ± 526,274	4097,631 ± 570,557
FAST	-	5630,432 ± 638,512	4753,127 ± 936,418
GFTT	-	1690,301 ± 134,873	826,943 ± 118,149
HARRIS	-	1631,748 ± 181,329	807,253 ± 119,363
KAZE	-	5495,831 ± 474,662	5009,814 ± 339,821
MSER	-	2393,933 ± 235,747	2184,669 ± 346,568
ORB	-	**392,888 ± 102,242**	679,487 ± 223,137

Tabela 15 – Tabela com as médias e desvios padrão em milissegundos dos tempos de execução dos algoritmos no LG Nexus 5, utilizando a correspondência por *BruteForce*

	AKAZE	*BRISK*	*ORB*
AGAST	-	11383,171 ± 1973,539	7444,114 ± 1070,341
AKAZE	3429,793 ± 503,015	2397,651 ± 544,356	2035,806 ± 314,051
BRISK	-	4594,276 ± 514,741	4434,278 ± 397,950
FAST	-	8784,020 ± 1360,668	6877,488 ± 801,806
GFTT	-	1772,006 ± 83,377	1334,291 ± 265,374
HARRIS	-	1725,925 ± 117,201	1255,975 ± 280,197
KAZE	-	5000,183 ± 661,745	4519,045 ± 489,394
MSER	-	2042,369 ± 172,110	2059,612 ± 232,375
ORB	-	**422,986 ± 128,531**	1203,980 ± 354,835

Tabela 16 – Tabela com as médias e desvios padrão em milissegundos dos tempos de execução dos algoritmos no LG Nexus 5, utilizando a correspondência por *BruteForce-Hamming*

	AKAZE	*BRISK*	*ORB*
AGAST	-	9122,201 ± 1441,930	5517,584 ± 1197,994
AKAZE	2574,919 ± 432,233	1751,017 ± 378,639	1465,602 ± 109,643
BRISK	-	3606,510 ± 524,496	3888,530 ± 559,868
FAST	-	6108,691 ± 692,254	4045,192 ± 885,750
GFTT	-	1627,473 ± 127,941	697,959 ± 121,205
HARRIS	-	1585,898 ± 180,240	691,657 ± 126,254
KAZE	-	4188,780 ± 503,241	3699,020 ± 153,967
MSER	-	1951,606 ± 219,144	1738,495 ± 327,349
ORB	-	**345,059 ± 98,131**	590,121 ± 221,748

Tabela 17 – Tabela com as médias e desvios padrão em milissegundos dos tempos de execução dos algoritmos no LG Nexus 5, utilizando a correspondência por *BruteForce-Hamming Look-up Table*

	AKAZE	*BRISK*	*ORB*
AGAST	-	8028,124 ± 1240,297	5668,650 ± 1213,363
AKAZE	2584,207 ± 433,757	1741,117 ± 381,273	1457,804 ± 113,025
BRISK	-	3622,069 ± 485,602	3867,581 ± 549,441
FAST	-	7127,632 ± 743,377	4031,857 ± 885,353
GFTT	-	1622,426 ± 131,100	697,100 ± 121,223
HARRIS	-	1579,200 ± 174,314	684,647 ± 126,213
KAZE	-	4189,412 ± 500,835	3725,809 ± 169,870
MSER	-	1950,383 ± 218,153	1740,805 ± 332,795
ORB	-	**345,549 ± 100,355**	589,940 ± 222,793

monstram as médias e desvios no tempo de execução no smartphone de modelo Motorola XT1097.

Tabela 18 – Tabela com as médias e desvios padrão em milissegundos dos tempos de execução dos algoritmos no Motorola XT1097, utilizando a correspondência por *BruteForce*

	AKAZE	*BRISK*	*ORB*
AGAST	-	8964,881 ± 1751,070	7552,155 ± 1021,874
AKAZE	3304,151 ± 486,767	2308,195 ± 526,174	1909,299 ± 305,501
BRISK	-	4442,368 ± 502,622	4264,701 ± 372,216
FAST	-	10538,903 ± 2410,340	7851,746 ± 1038,890
GFTT	-	1695,103 ± 80,046	1284,957 ± 254,866
HARRIS	-	1656,068 ± 119,187	1200,701 ± 268,157
KAZE	-	5458,175 ± 755,097	4942,312 ± 874,129
MSER	-	2065,586 ± 230,974	2018,171 ± 218,557
ORB	-	**412,066 ± 122,464**	1174,133 ± 342,510

Esse método de cálculo de tempo por iteração do algoritmo foi usado, pois o tempo que o método que computa a matriz de transformação a partir de um conjunto de características coincidentes leva para executar, aumenta significativamente conforme a quantidade de pontos correspondentes na imagem for aumentada. Dessa forma, algoritmos que detectam muitos pontos indiscriminadamente, mesmo fazendo isso rapidamente, têm tempos de execução altos ao final. Os tempos apresentados na Análise 2 são superiores à Análise 1 pois incluem o tempo de processamento do cálculo da matriz de transformação e a resolução dos quadros de imagem, fornecida pela base KITTI, utilizada é maior que os quadros da Análise 1. A tabela 21 demonstra a quantidade média de *features* capturadas durante a execução dos testes nos smartphones.

Para que se pudesse avaliar o desempenho dos algoritmos no smartphone com relação ao computador, os mesmos testes foram executados em um computador Quad-Core

Tabela 19 – Tabela com as médias e desvios padrão em milissegundos dos tempos de execução dos algoritmos no Motorola XT1097, utilizando a correspondência por *BruteForce-Hamming*

	AKAZE	*BRISK*	*ORB*
AGAST	-	6457,558 ± 1022,802	5241,235 ± 1262,093
AKAZE	2509,455 ± 431,120	1684,874 ± 370,544	1394,529 ± 111,272
BRISK	-	3500,348 ± 519,391	3713,362 ± 522,817
FAST	-	5722,914 ± 778,138	4617,799 ± 1043,444
GFTT	-	1584,041 ± 131,579	674,382 ± 118,936
HARRIS	-	1544,801 ± 187,570	657,654 ± 131,215
KAZE	-	4737,821 ± 672,677	4028,860 ± 256,914
MSER	-	1947,661 ± 215,995	1711,674 ± 322,709
ORB	-	**342,457 ± 96,366**	571,862 ± 218,648

Tabela 20 – Tabela com as médias e desvios padrão em milissegundos dos tempos de execução dos algoritmos no Motorola XT1097, utilizando a correspondência por *BruteForce-Hamming Look-up Table*

	AKAZE	*BRISK*	*ORB*
AGAST	-	6518,952 ± 834,380	5156,329 ± 1106,533
AKAZE	2464,474 ± 425,057	1688,550 ± 365,147	1400,481 ± 105,799
BRISK	-	3495,588 ± 521,292	3705,809 ± 518,425
FAST	-	5747,689 ± 784,375	4844,121 ± 1122,280
GFTT	-	1581,603 ± 136,771	674,060 ± 121,256
HARRIS	-	1546,916 ± 185,542	658,593 ± 124,246
KAZE	-	4690,724 ± 869,586	4083,178 ± 290,205
MSER	-	1915,642 ± 212,933	1717,612 ± 327,600
ORB	-	**342,622 ± 98,033**	575,316 ± 210,166

Tabela 21 – Tabela com a quantidade média de *features* capturadas durante a execução dos algoritmos nos smartphones

	AKAZE	*BRISK*	*ORB*
AGAST	-	7016,960	6176,560
AKAZE	2268,450	2268,450	2259,270
BRISK	-	4067,230	3945,520
FAST	-	7446,960	6609,300
GFTT	-	967,650	914,620
HARRIS	-	969,240	918,990
KAZE	-	2381,320	2247,400
MSER	-	365,980	380,310
ORB	-	403,390	500,000

2.4GHz Intel-Core i7 com 12 GB de *RAM* e executando o SO Microsoft Windows 10. Os resultados das médias e desvios dos tempos de execução para os algoritmos *BruteForce*; *BruteForce-Hamming* e *BruteForce-Hamming Look-up Table* estão dispostos nas tabelas 24; 25 e 26 respectivamente, que se encontram no apêndice A deste trabalho.

A Tabela 22 resume e mostra para cada detector utilizado a sua combinação mais rápida (também dada com os tempos de execução, erros e desvio padrão médios por quadro). A combinação mais rápida foi: detector *ORB*, descritor *BRISK* e força bruta *Hamming* com *LUT*. Dado que o hardware de um smartphone é inferior ao computador, é esperado que seus tempos sejam mais elevados, ainda assim, os tempos apresentados nos smartphones são tempos aceitáveis para uma aplicação que permita uma demora de 350 ms para a execução do passo da odometria visual.

Tabela 22 – Combinações mais rápidas de cada detector com seus tempos de execução, erros e desvio padrão em relação à pose real

Detector	Descritor	Comparador	Tempo Computador (ms)	Tempo Moto XT1097 (ms)	Tempo Galaxy (ms)	Tempo Nexus (ms)	Erro (m)	Desvio Padrão
ORB	*BRISK*	*LUT*	144,234	342,457	392,441	392,441	2,962	1,883
HARRIS	*ORB*	*LUT*	240,863	1544,800	1635,955	1585,897	2,545	0,998
GFTT	*BRISK*	*LUT*	265,073	4617,798	4571,191	4045,192	2,175	1,930
MSER	*ORB*	*LUT*	326,735	1711,674	2184,063	1738,494	1,833	0,687
AKAZE	*ORB*	*BFH*	304,148	1688,550	2009,209	1741,117	2,085	0,899
KAZE	*ORB*	*BFH*	657,764	4083,178	5009,813	3725,808	2,204	0,928
BRISK	*ORB*	*LUT*	1482,935	3500,347	3599,680	3606,509	2,236	0,899
FAST	*BRISK*	*LUT*	1634,549	3713,362	4105,163	3888,529	2,343	1,127

A implementação usada da odometria visual guarda as informações referentes à pose calculada. A mesma é então comparada com a pose real e, dado o resultado, tira-se a média relativa de erro e seu desvio padrão.

O teste para calcular a precisão da odometria visual foi feito a partir dos 300 quadros previamente selecionados. É importante realçar que a trajetória tem duas curvas (uma para a direita e outra para a esquerda), o que fornece uma avaliação satisfatória para verificar a propagação de erro.

As Figuras 25, 26, 27, 28, 29, 30 e 31 mostram as trajetórias de cada combinação possível entre os detectores, descritores e comparadores agrupadas por descritor e comparador.

Nesse teste, como mostra a Figura 25, a combinação de algoritmos mais precisa é: detector *HARRIS*, descritor *BRISK* e comparador força bruta euclidiano, com uma média de erro de 1,778 metros e desvio padrão de 0,475 metros. A Tabela 23 mostra as cinco combinações mais precisas (que também estão nas imagens citadas) com seus tempos de execução, erros e desvio padrão médios por quadro, por cada *hardware* testado.

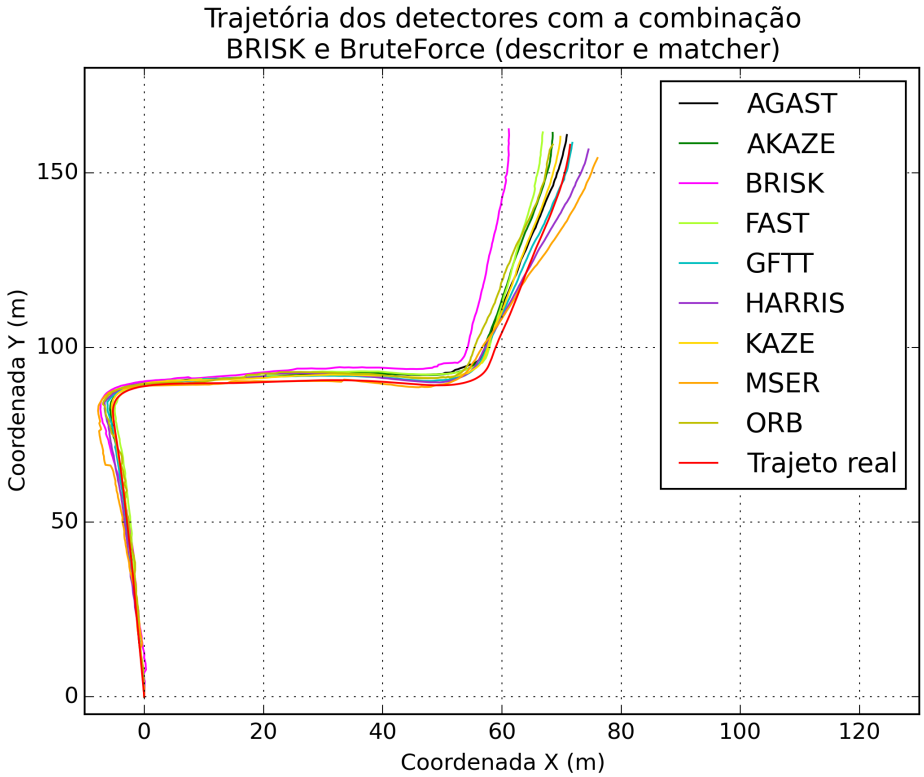


Figura 25 – Trajetórias utilizando o descritor *BRISK* e o comparador *BruteForce*.

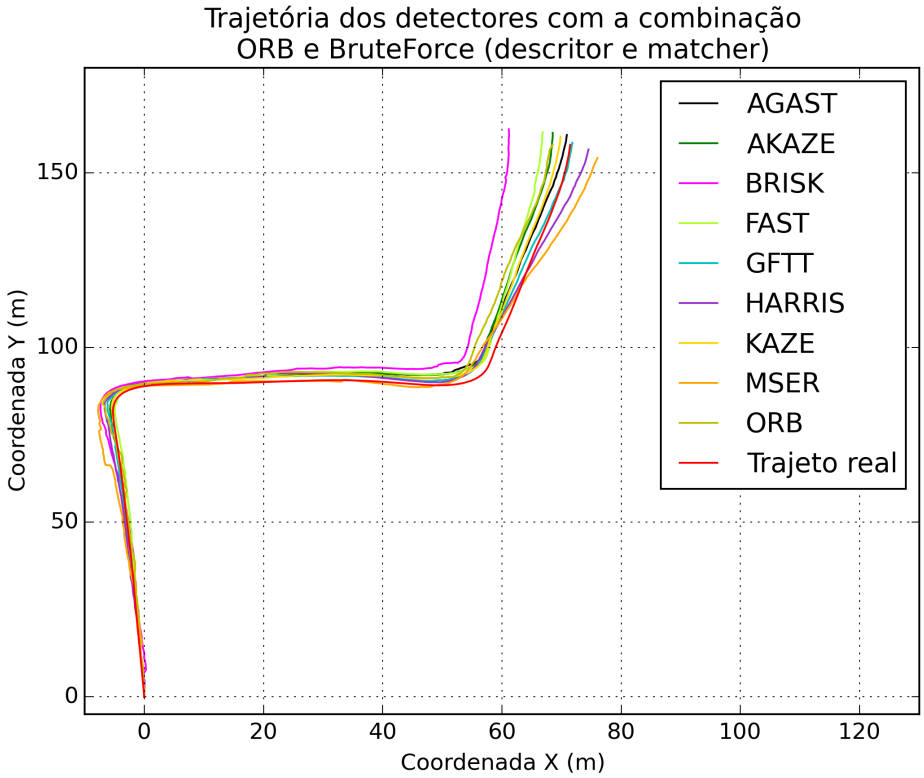


Figura 26 – Trajetórias utilizando o descritor *ORB* e o comparador *BruteForce*.

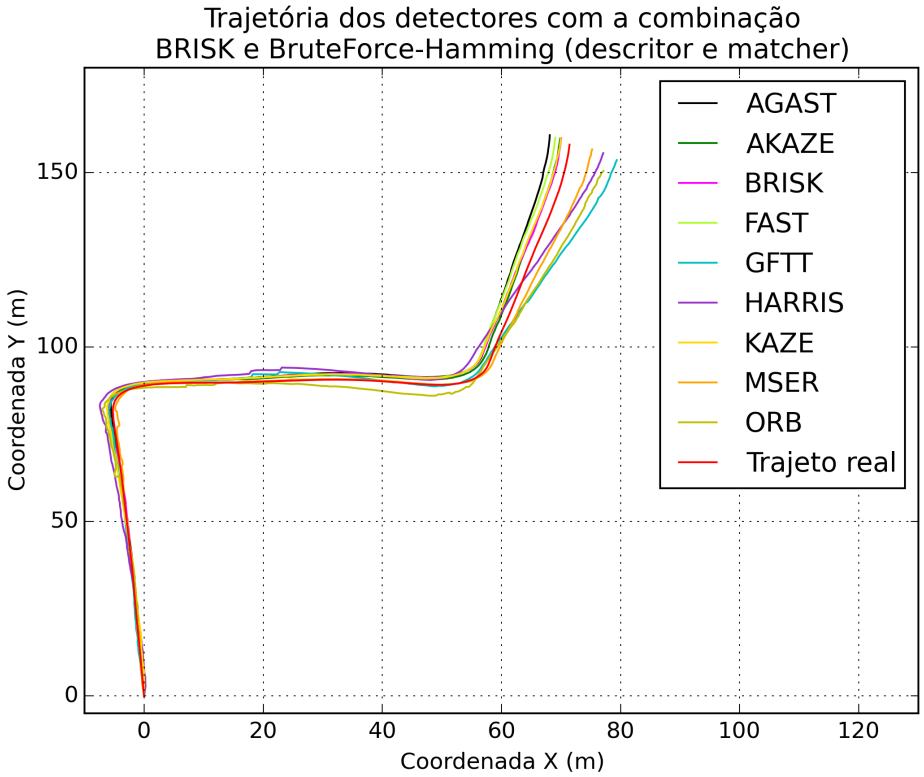


Figura 27 – Trajetórias utilizando o descritor *BRISK* e o comparador *BruteForce-Hamming*.

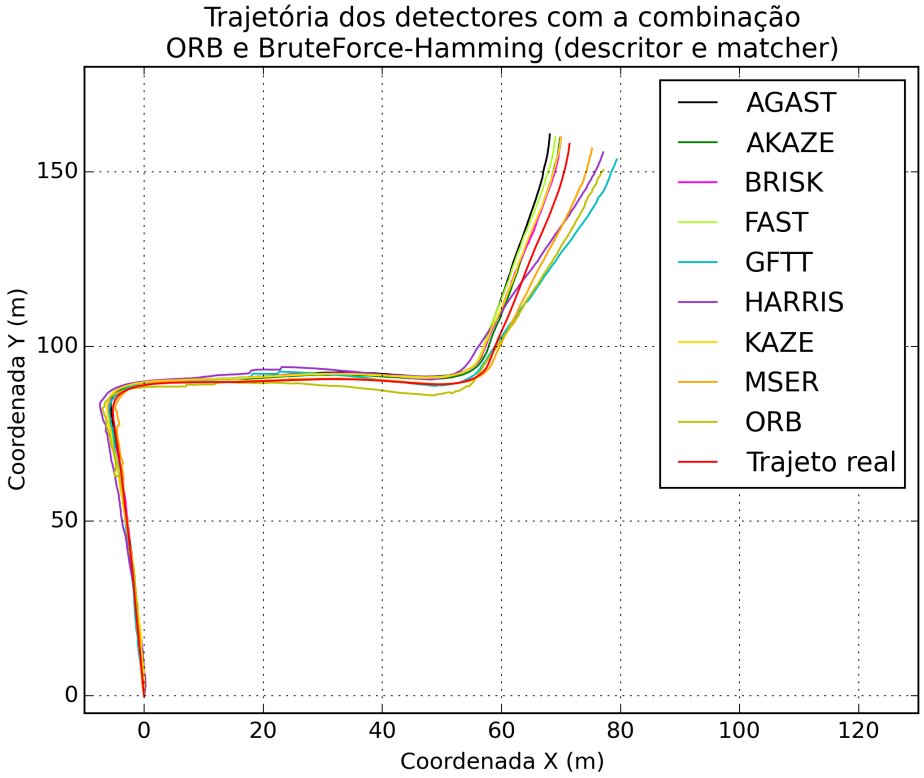


Figura 28 – Trajetórias utilizando o descritor *ORB* e o comparador *BruteForce-Hamming*.

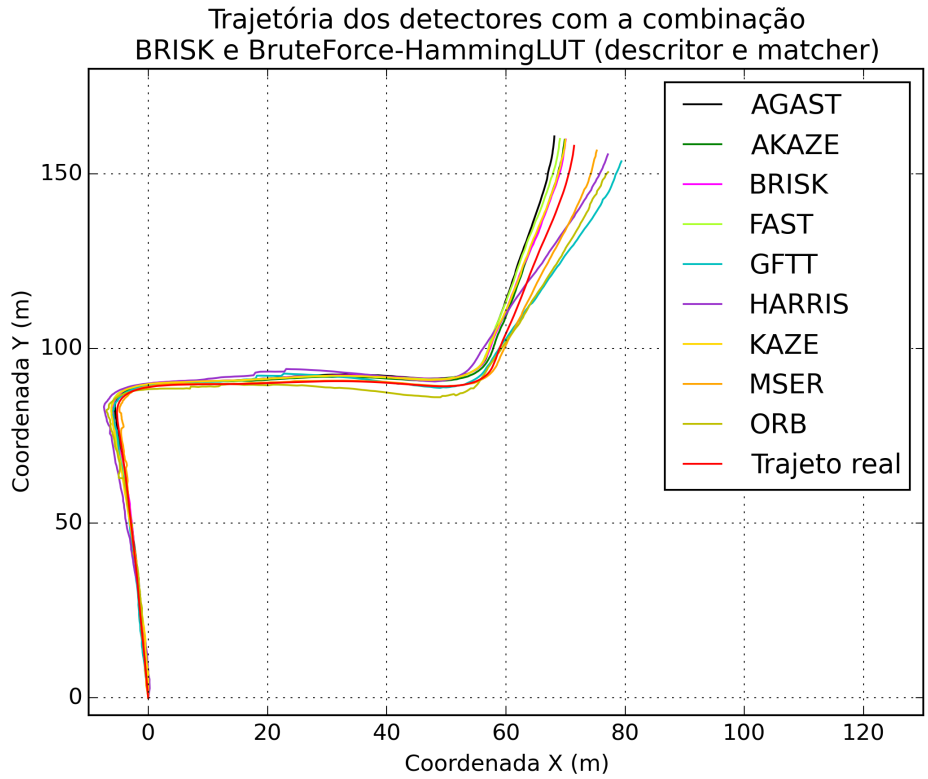


Figura 29 – Trajetórias utilizando o descritor *BRISK* e o comparador *BruteForce-Hamming LUT*.

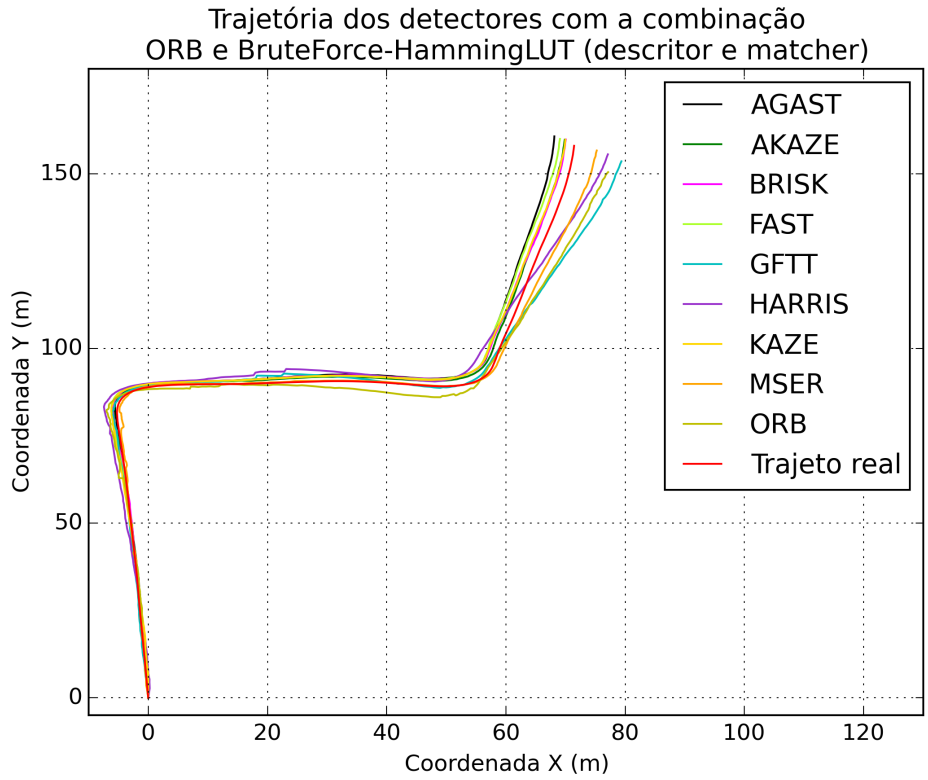


Figura 30 – Trajetórias utilizando o descritor *ORB* e o comparador *BruteForce-Hamming LUT*.

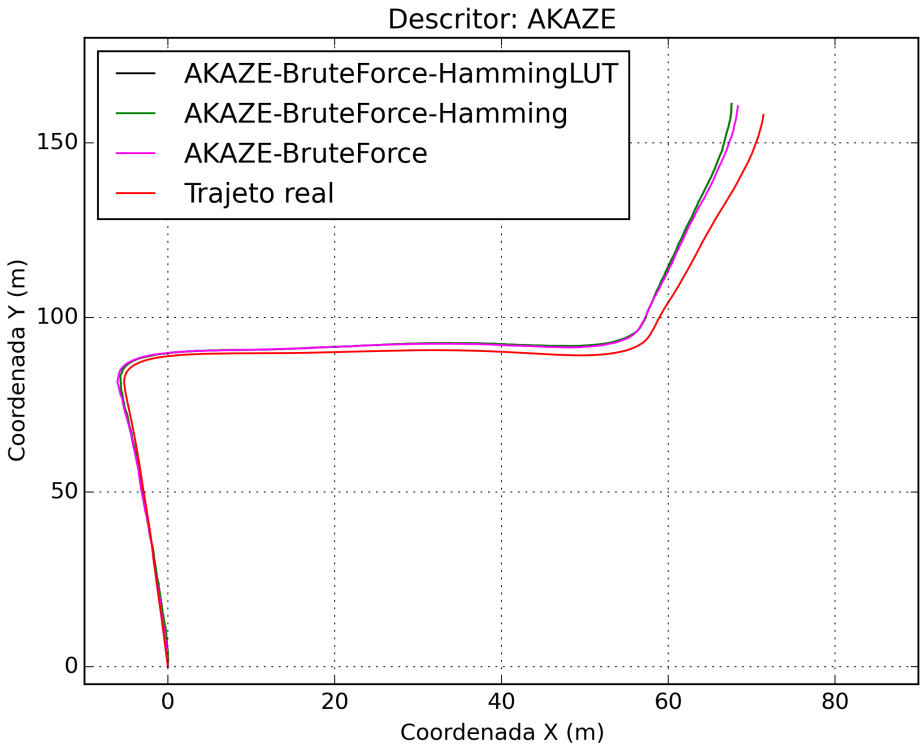


Figura 31 – Trajetórias utilizando o descritor e detector *AKAZE*.

Tabela 23 – Combinações mais precisas com seus tempos de execução, erros e desvio padrão em relação à pose real

Detector	Descritor	Comparador	Tempo Moto XT1097 (ms)	Tempo SM-T520 (ms)	Tempo Nexus 5 (ms)	Erro (m)	Desvio Padrão
HARRIS	*BRISK*	*BF*	1656,067	1771,899	1725,924	1,778	0,475
HARRIS	*ORB*	*BF*	1200,700	1355,365	1255,975	1,778	0,475
GFTT	*ORB*	*BF*	1284,957	1454,202	1334,291	1,787	0,573
GFTT	*BRISK*	*BF*	1656,067	1800,127	1725,924	1,787	0,573
MSER	*ORB*	*BFH*	1711,674	2184,063	1738,494	1,833	0,687

4.3 Cenários de Localização

Para a análise do sistema de localização foram considerados três cenários para verificar o tempo e precisão do sistema. Os cenários foram construídos na ferramenta de simulação V-rep, onde o sistema de localização obtém os dados simulados se conectando por meio de uma *API* remota do V-rep, que é executada em um computador.

A comunicação ocorre por meio de *Ethernet over USB* (*Universal Serial Bus*), que através do protocolo *CDC-EEM* permite emulação de uma interface *Ethernet*, utilizando o meio físico *USB*, diretamente do smartphone com computador onde o V-rep é executado. O simulador fornece as medidas de orientação do sensor *IMU*, leituras de sinal *RSS* e as imagens necessárias para a odometria visual. As medidas obtidas no simulador incluem a

modelagem dos erros de leitura do respectivo sensor.

4.3.1 Cenário 1

O primeiro cenário é composto por uma sala de cantos arredondados, contendo 3 *APs WiFi* (Figura 32). Neste cenário foi analisada a influência da quantidade de *features* detectadas na odometria visual. O robô realizou 4220 observações do ambiente ao longo do deslocamento e os resultados de precisão e tempo de execução foram analisados.

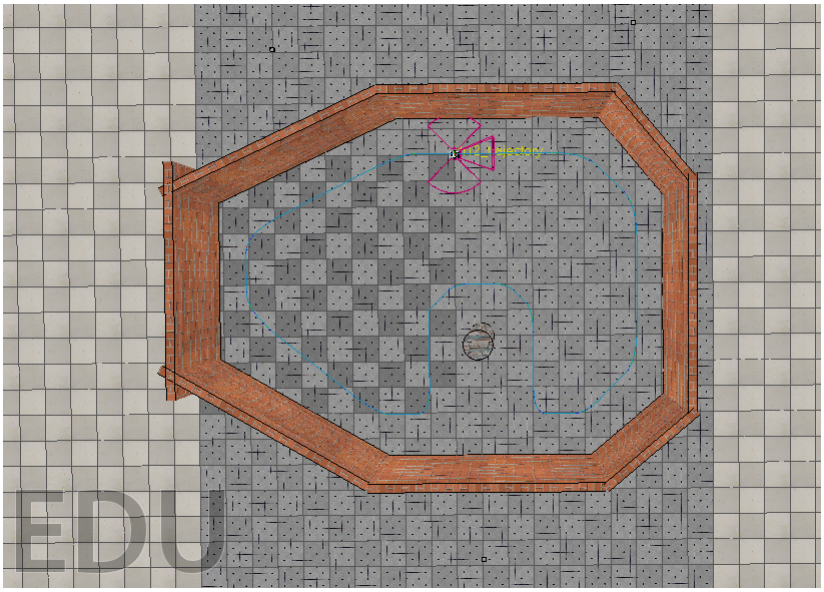


Figura 32 – Imagem do ambiente utilizado para os testes do cenário 1.

A Figura 33 mostra as trajetórias realizadas pelo robô em relação à trajetória real percorrida: uma executada somente com odometria visual e outra executada pela odometria visual corrigida pelo filtro de partículas com *WiFi* com variação na quantidade de *features*. A Figura 34 mostra um gráfico contendo as médias e desvios do erro quadrático das trajetórias realizadas. O gráfico revela que a quantidade de *features* não causará um aumento do erro na pose do robô após a aplicação do filtro de partículas.

As Figuras 35, 36 e 37 mostram os tempos de execução do algoritmo de odometria visual com relação à quantidade média de *features* detectadas nos dispositivos móveis Nexus 5, XT 1097 e SM-T520 respectivamente. Os gráficos mostram que o tempo de execução, em todos os dispositivos, cresce à medida que se aumenta a quantidade de *features*.

4.3.2 Cenário 2

O segundo cenário analisado acontece em mapa que modela, em escala real, o mapa do Departamento de Computação (DC/UFPI) (Figura 38). Neste cenário foi verificado os resultados da variação da quantidade de partículas no filtro. Nos testes foram utilizadas

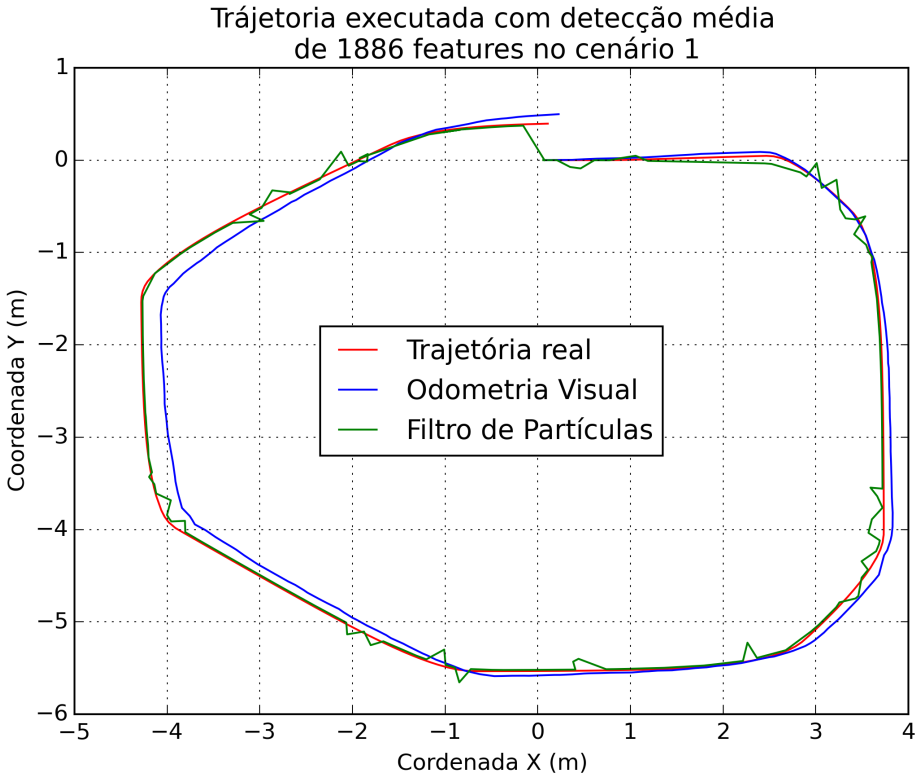


Figura 33 – Comparação da trajetória do robô utilizando uma média de 824 *features*.

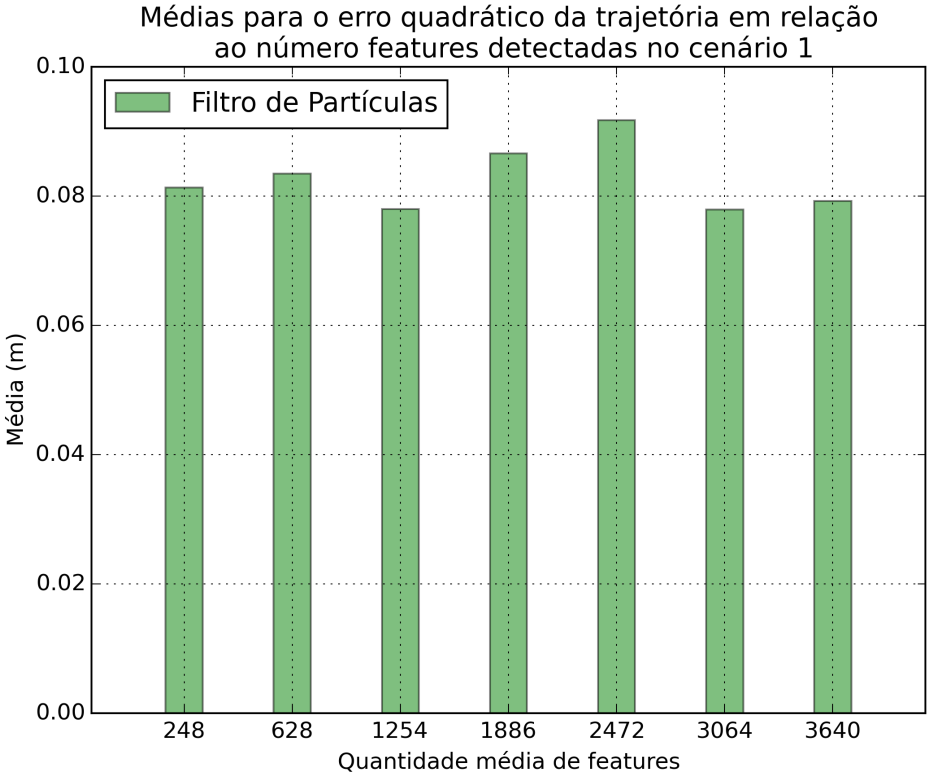


Figura 34 – Gráfico das médias do erro quadrático da trajetória no primeiro cenário.

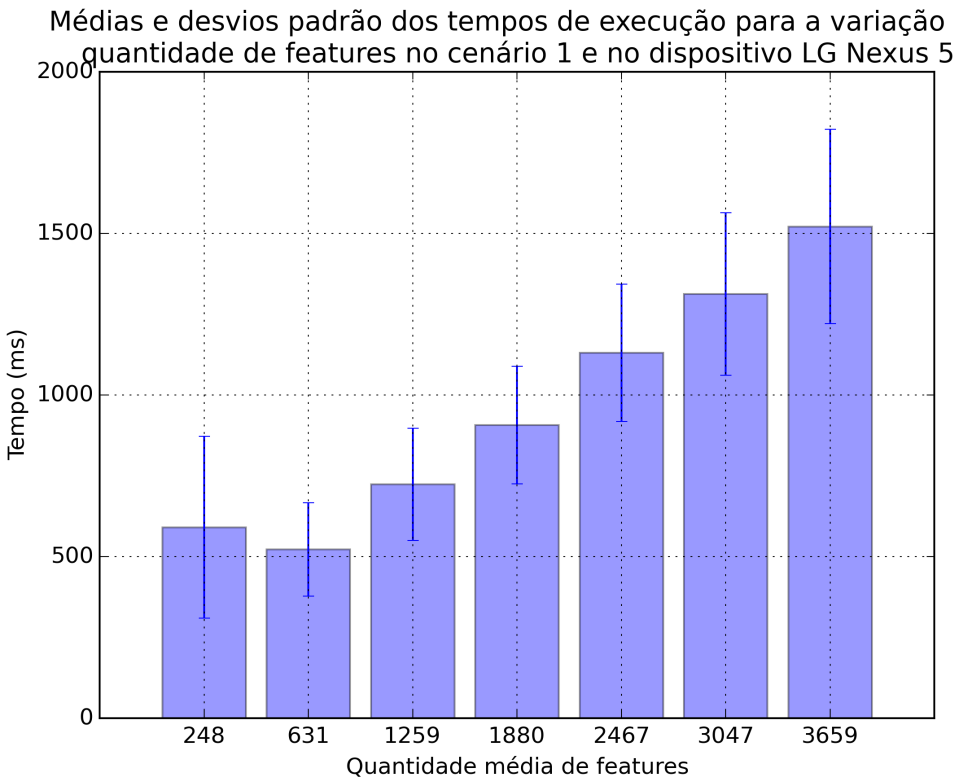


Figura 35 – Gráfico das médias e desvios do erro quadrático da trajetória no primeiro cenário.

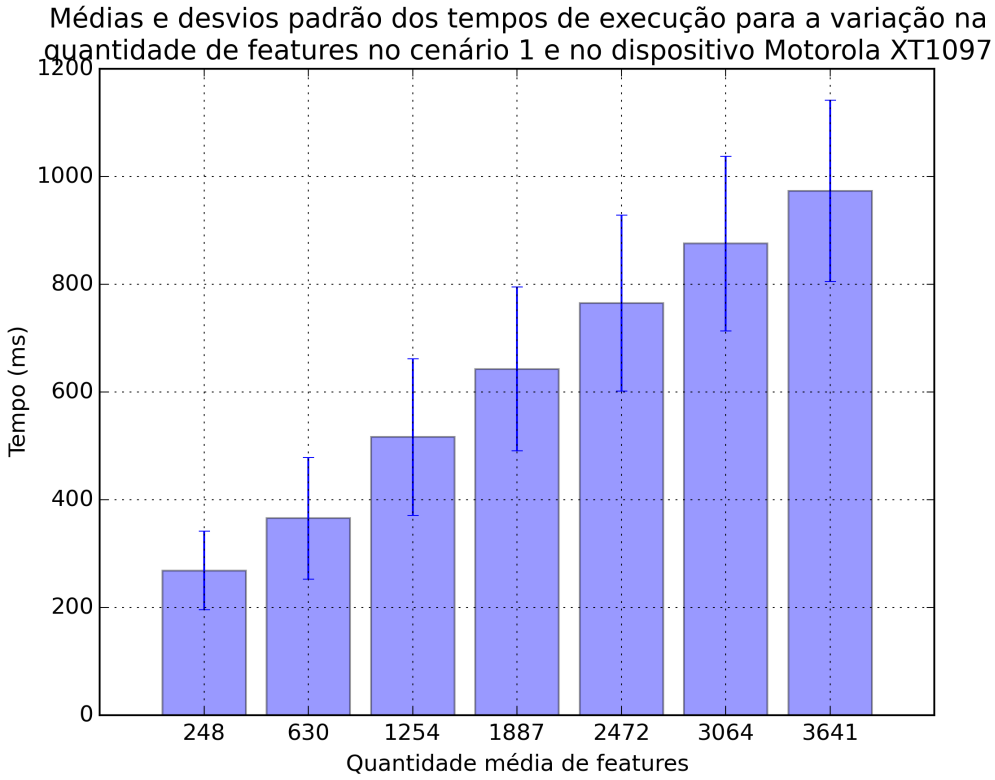


Figura 36 – Gráfico das médias e desvios do erro quadrático da trajetória no primeiro cenário.

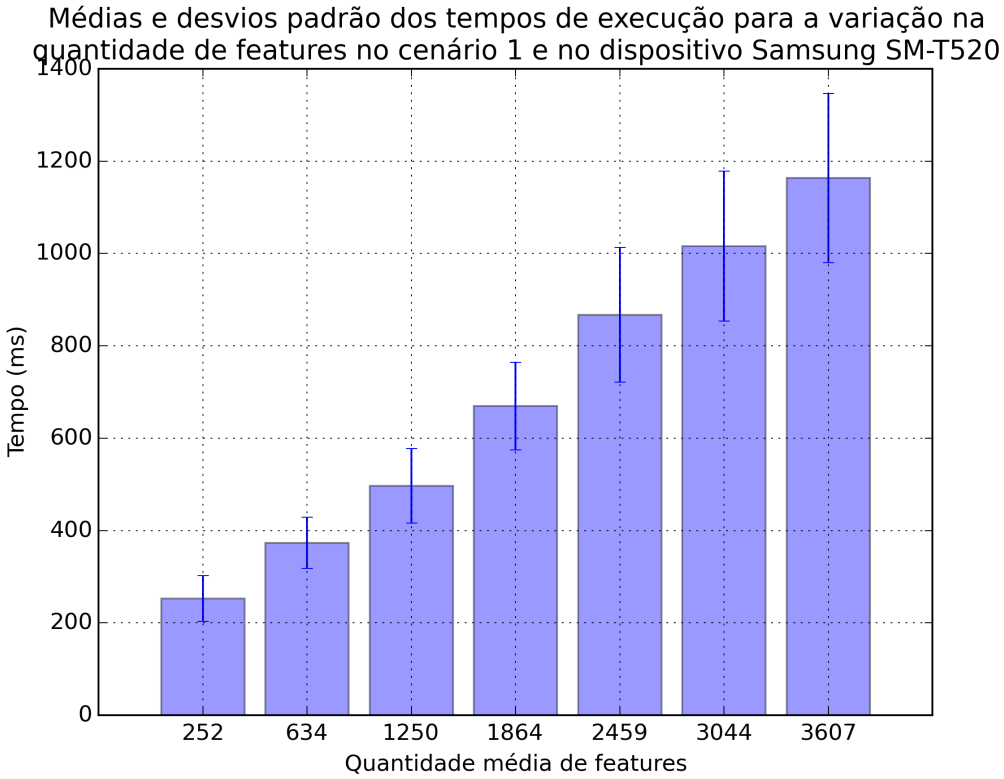


Figura 37 – Gráfico das médias e desvios do erro quadrático da trajetória no primeiro cenário.

as quantidades de 250, 500, 750, 1000, 1250, 1500 e 1750 partículas e foram feitas 3650 observações.

A Figura 39 mostra as trajetórias realizadas pelo robô em relação a trajetória real percorrida: uma executada somente com odometria visual e outra executada pela odometria visual corrigida pelo filtro de partículas com *WiFi* com variação na quantidade de *features*. No início do trajeto, como visto na Figura 39, a localização do filtro de partículas diverge bastante da trajetória real, pois a mesma ainda não convergiu para uma localização próxima a real. 40 mostra um gráfico contendo as médias e desvios do erro quadrático das trajetórias realizadas. O gráfico revela que um aumento na quantidade de partículas não traz uma grande melhora na precisão da localização, a variação de 250 a 1500 partículas se manteve com erro próximo a 1,0 metro.

As Figuras 41, 42, 43 mostram gráficos com a média e desvio, em milissegundos, do tempo de execução do filtro de partículas em cada dispositivo, variando a quantidade de partículas utilizadas. Os gráficos mostram que o aumento na quantidade de partículas causa uma elevação no custo computacional de execução do sistema.

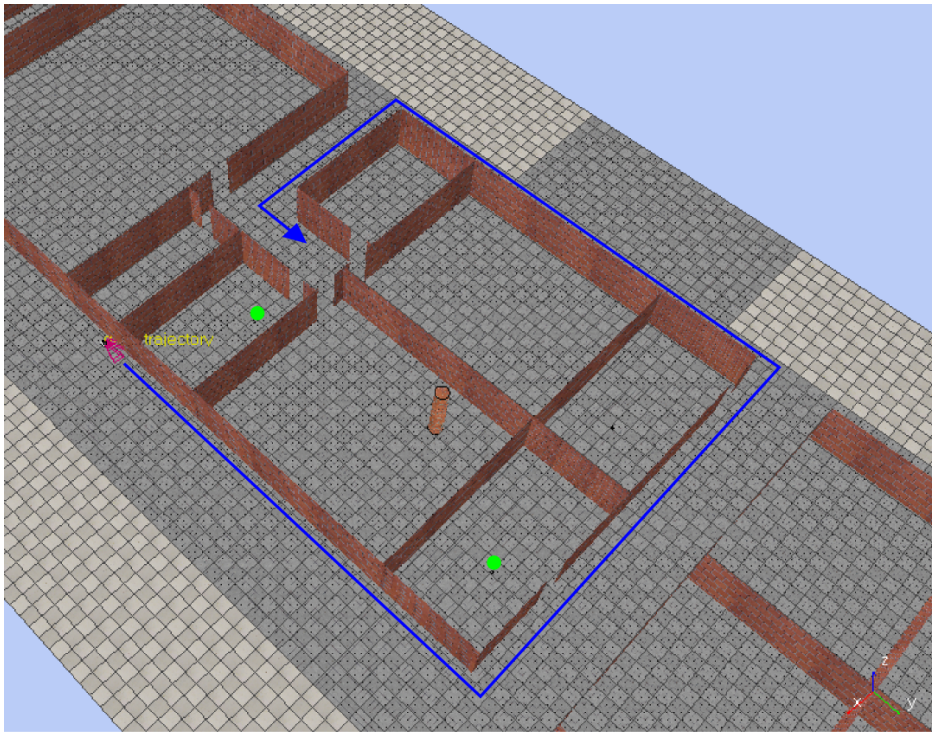


Figura 38 – Imagem do mapa do Departamento de Computação, ambiente utilizado para os testes do cenário 2.

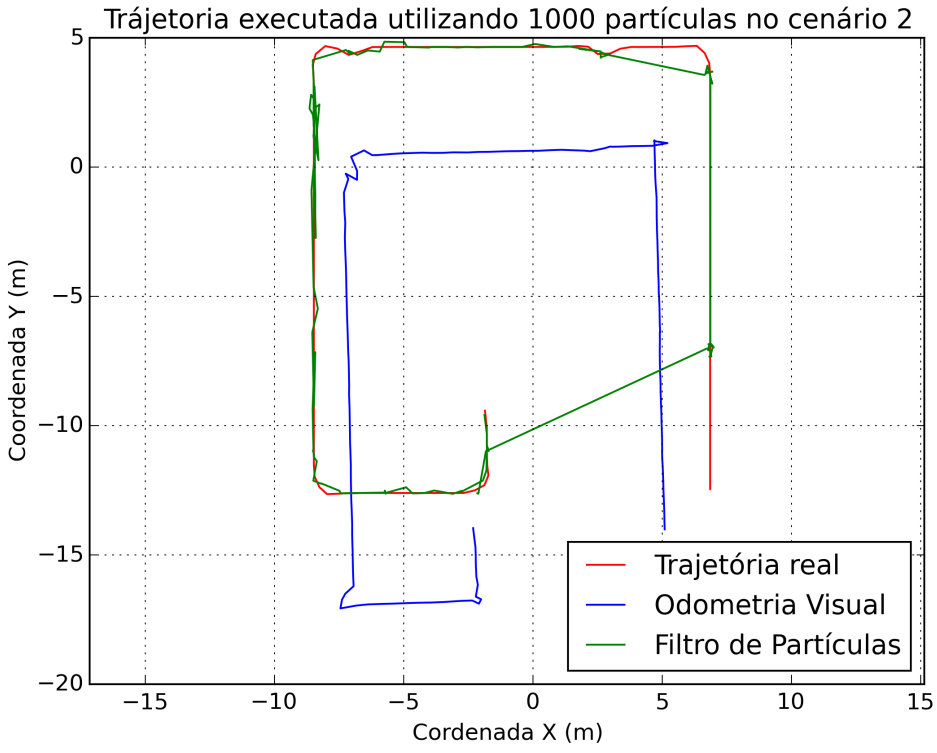


Figura 39 – Comparação da trajetória do robô utilizando 1000 partículas.

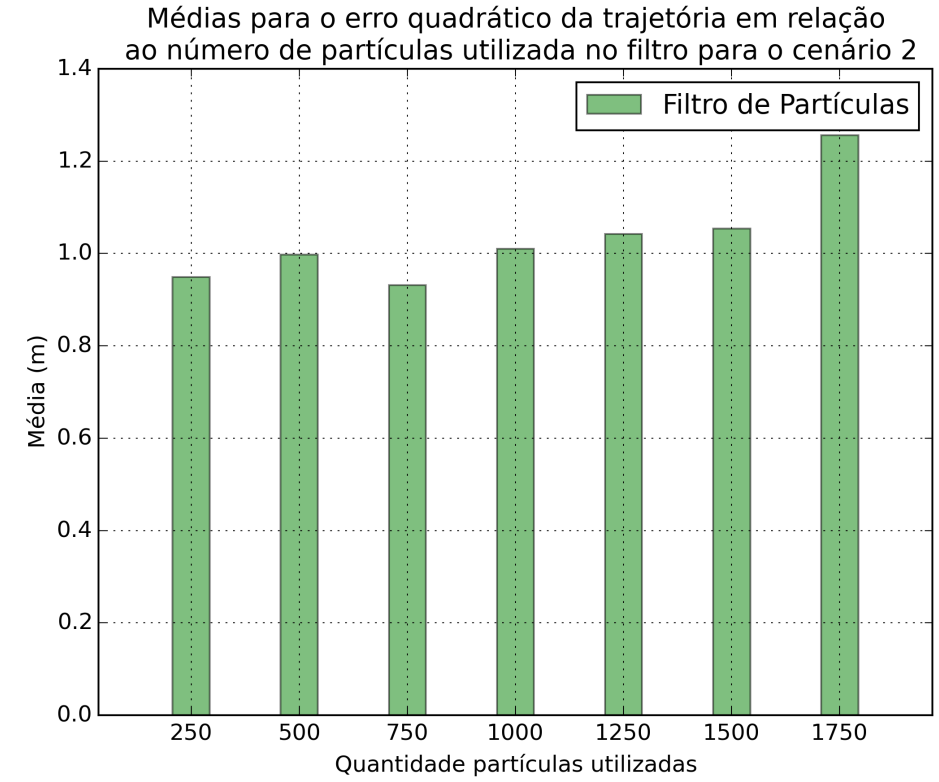


Figura 40 – Médias para o erro quadrático da trajetória variando a quantidade de partículas.

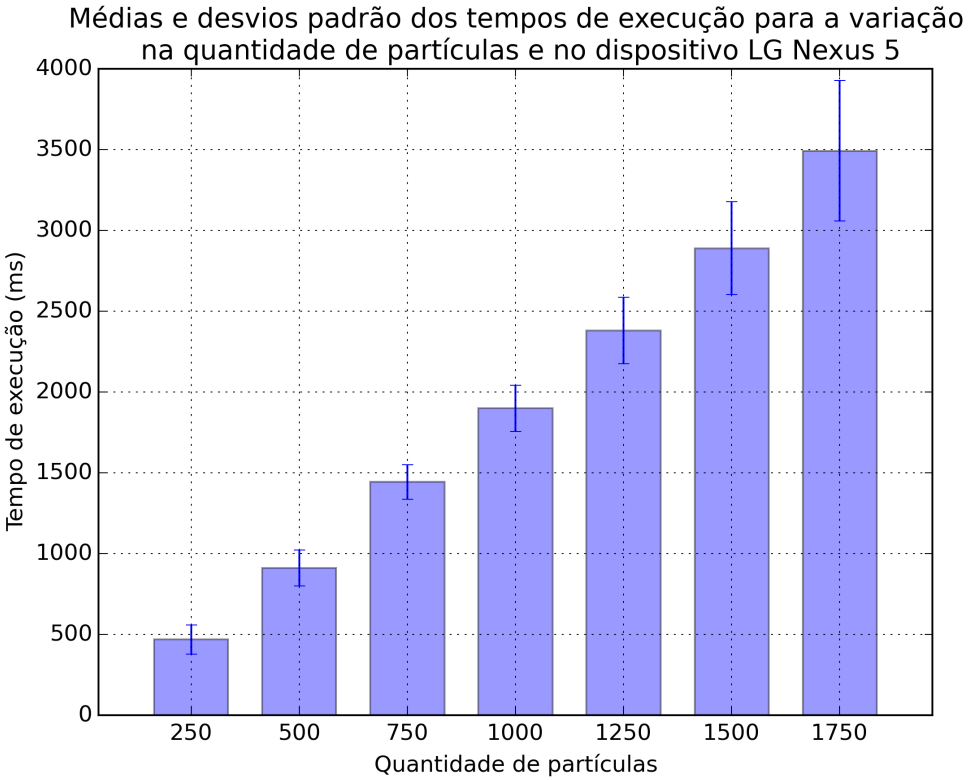


Figura 41 – Médias e desvio dos tempos de execução do filtro variando a quantidade de partículas no smartphone LG Nexus 5.

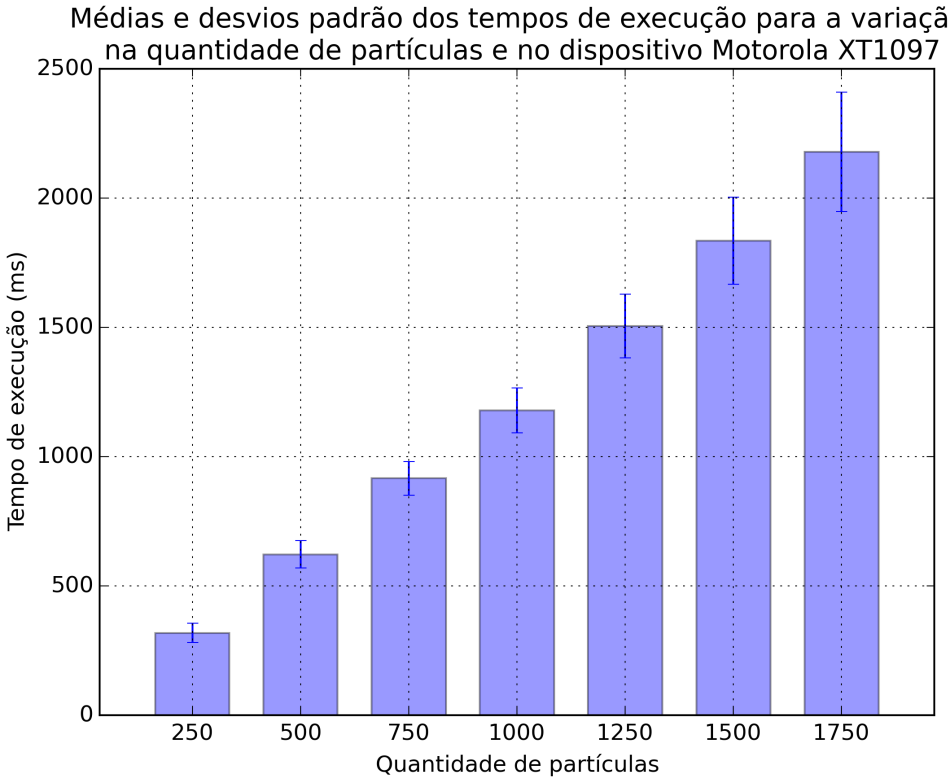


Figura 42 – Médias e desvio dos tempos de execução do filtro variando a quantidade de partículas no smartphone Motorola XT1097.

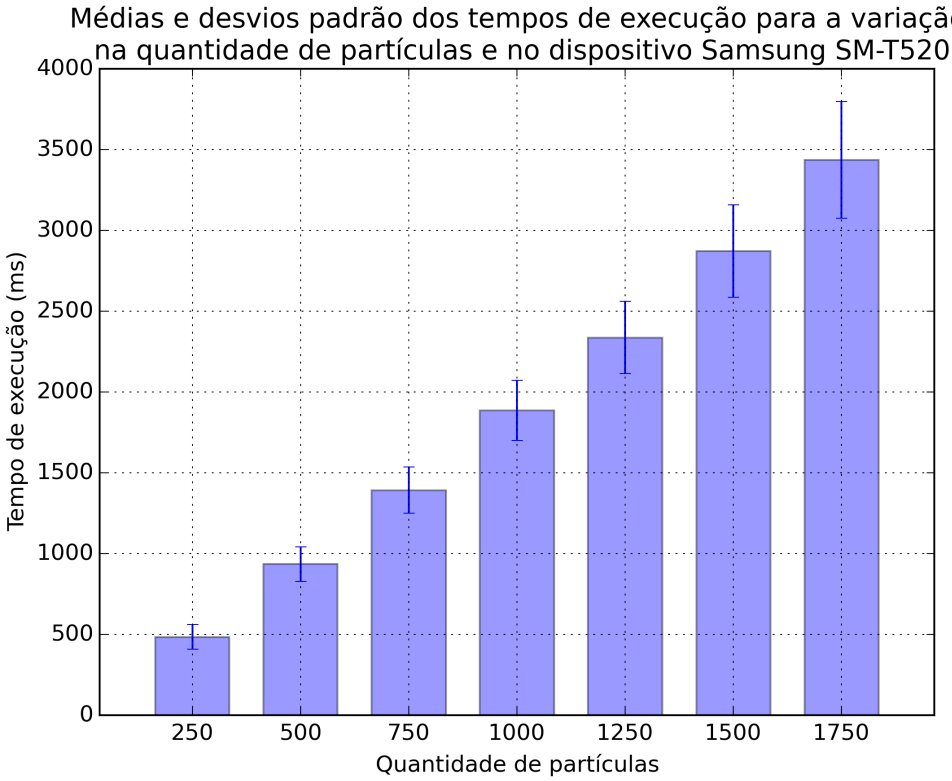


Figura 43 – Médias e desvio dos tempos de execução do filtro variando a quantidade de partículas no smartphone Samsung SM-T520.

4.3.3 Cenário 3

Dado que o erro da localização durante a variação de partículas mostrou ser em torno de 2 metros, foi escolhido um terceiro cenário para avaliação do sistema. O terceiro cenário consiste na simulação de um robô cadeira de rodas deslocando-se por uma ala hospitalar inspirada no Hospital Universitário da UFPI. A escolha desse cenário foi motivada pelo fato de que o aumento das dimensões do robô causam uma redução no erro de localização. As dimensões de uma cadeira de rodas padrão são de 1,15 x 0,70 x 0,46 metros (largura x comprimento x altura), segundo a ("NBR9050", "2015") e como visto na Figura 44.

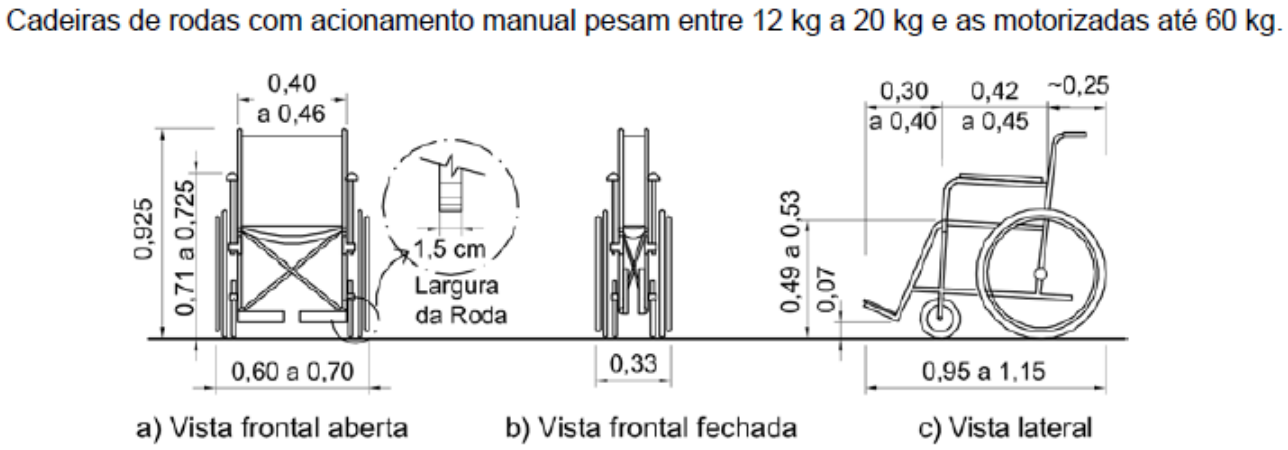


Figura 44 – Dimensões padrões de uma cadeira de rodas segundo a "NBR9050" ("2015").

O cenário foi modelado no V-rep (Figura 38) como um conjunto de corredores e quartos de pacientes. Neste cenário foram feitas 4300 observações e analisou-se a influência da quantidade de *APs* na precisão da localização. Neste teste foi utilizado as quantidades de 1, 3, 5 e 10 *APs* espalhados pelo mapa.

Assim como nos cenários anteriores, comparou-se a trajetória do robô com as trajetórias da odometria e do filtro como visto na Figura 46. A Figura 47 mostra o resultado das médias e desvio do erro quadrático das trajetórias executadas. Verificou-se que com uma quantidade superior a 3 *APs* não houve uma melhora significativa na precisão da localização.

No Apêndice, nas Figuras 48, 49, 50, 53, 51 e 52 encontram-se os resultados de trajetória para cada variação da quantidade de *features*. As Figuras 54, 55, 56, 57, 58 e 59 mostram os demais resultados de trajetória para as variações da quantidade de partículas. As Figuras 60, 61, 62 e 63 representam as trajetórias executadas pelo robô variando a quantidade de *APs*.

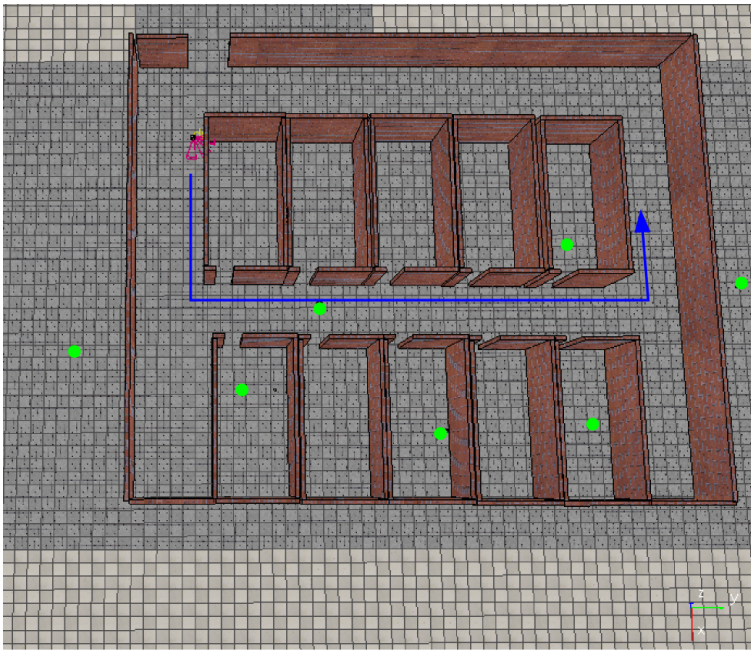


Figura 45 – Imagem que demonstra o ambiente do terceiro cenário, uma ala hospitalar.

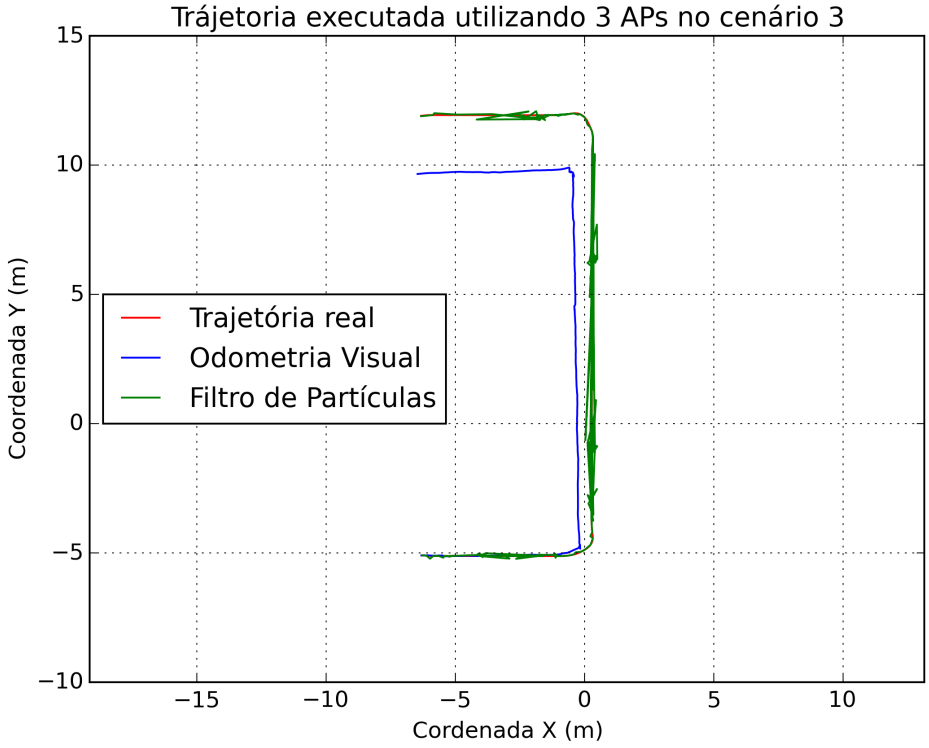


Figura 46 – Comparação da trajetória do robô utilizando 3 *Access Points*.

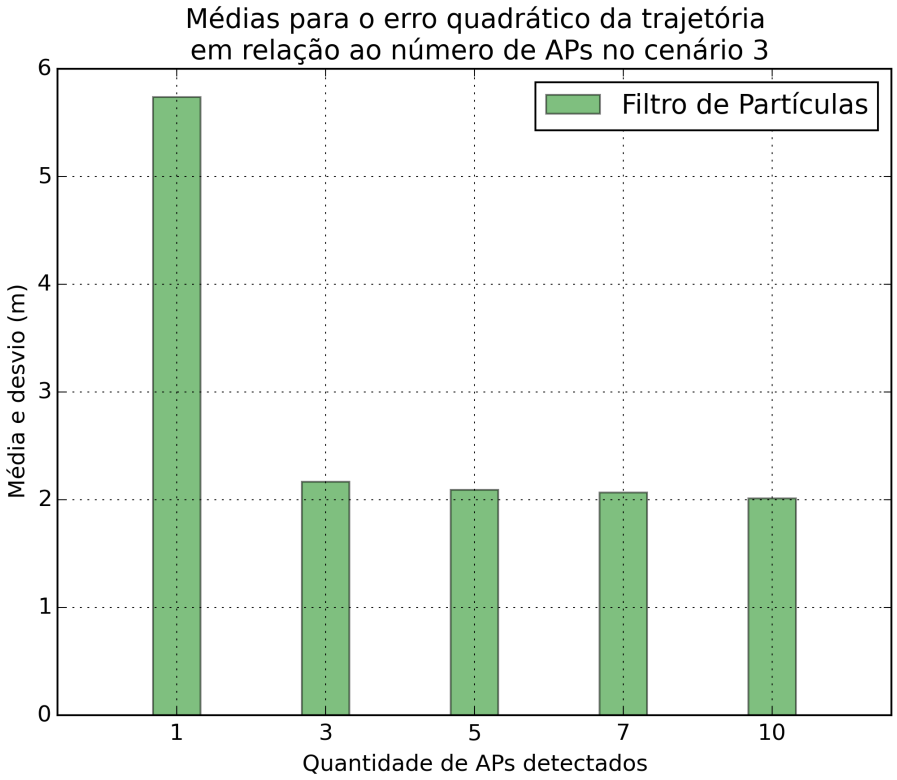


Figura 47 – Médias para o erro quadrático da trajetória variando a quantidade de *APs*.

5 Conclusões e Trabalhos Futuros

Este trabalho apresentou a proposta de um sistema de localização para *Cellbots* utilizando sensores presentes no dispositivo móvel. A utilização do *cellbot* provê um plataforma de baixo custo e de hardware de fácil acesso para aplicações robóticas. Neste trabalho é apresentado a utilização de um sistema de odometria visual, um sistema de localização *WiFi* por analise de cenário e um filtro de partículas todos implementados para smartphone demonstrando a capacidade e potencial da arquitetura *cellbot*. A fusão dos três sistemas citados compõem o sistema de localização proposto neste trabalho.

O sistema realiza a localização do *cellbot* através da estimativa do deslocamento por odometria visual, estimativa de pose utilizando *RSS WiFi* e a correção da pose ocorre com a utilização do filtro probabilístico de partículas. As imagens para a realização da *VO* e sinal dos *AP*s *WiFi* são obtidas e processadas pelo próprio dispositivo móvel do *cellbot*, assim como todo o processamento do sistema de localização.

Foram realizadas análises com dados reais e dados simulados com relação ao custo computacional e precisão de trajetória dos algoritmos envolvidos na odometria visual (detecção, descrição, correspondência e cálculo da matriz de transformação), desempenho do filtro de partículas (variação na quantidade de partículas e pontos de acesso *WiFi*) no robô com dispositivo móvel e os valores são apresentados através de gráficos e tabelas. Os resultados das análises revelaram que o sistema é capaz de realizar a localização de um robô móvel. As análises da precisão entre trajetória real executada pelo robô e a obtida pelo robô mostraram uma margem de erro próximo a 2,0 metros que pode ser considerada aceitável dependendo do tamanho do robô utilizado. No cenário 3 foi proposto a utilização do sistema em um robô cadeira de rodas, que possui dimensões próximas a 1,0 metro quadrado, e o sistema foi capaz de determinar a pose do robô.

Quanto ao custo computacional dos algoritmos utilizados, o sistema se mostrou aceitável dependendo da aplicação. Com a utilização de uma média de 1200 *features* na odometria visual e 750 partículas no filtro, o sistema de localização leva aproximadamente 1,9 segundos para executar cada ciclo do processo de localização. E assim, não se aplica a cenários onde o robô pretende realizar tarefas em velocidades altas, funcionando melhor em sistemas que operam em velocidade mais reduzidas. Os resultados mostrados revelaram que o sistema é eficaz para a localização de um robô cadeira de rodas em um ambiente hospitalar.

Como trabalhos futuros, planeja-se:

- Melhorar a eficiência do sistema de localização realizando uma implementação que

faça uso do poder de multiprocessamento fornecida pelos *smartphones* realizando a paralelização dos algoritmos utilizados no sistema;

- Realizar uma análise com outros algoritmos de processamento de imagem (Fluxo óptico), estratégias de estimativa de pose e filtros probabilísticos em busca de um melhor desempenho do sistema de localização;
- Integrar o sistema de localização em um sistema de navegação completo, adicionando as etapas de mapeamento, planejamento e execução de trajetória;
- Realizar localização e mapeamento simultâneos (*Simultaneous localization and mapping*, *SLAM*), que se trata de realizar a criação do mapa do ambiente e encontrar a localização do robô móvel neste mapa à medida que se navega no ambiente.

Por fim, a etapa de análise da odometria visual em smartphones deste trabalho gerou produção científica no evento *ICIAP* 2016 - *18th International Conference on Image Analysis and Processing* intitulado *Comparison and Effectiveness of Visual Features Detectors and Binary Descriptors for a Monocular Visual Odometry System Using Cellbots.*

Referências

AGRAWAL, M.; KONOLIGE, K. Censure: Center surround extremas for realtime feature detection and matching. In: *ECCV*. [S.l.: s.n.], 2008. Citado na página 26.

ALAHI, A.; ORTIZ, R.; VANDERGHEYNST, P. Freak: Fast retina keypoint. In: *CVPR*. IEEE Computer Society, 2012. p. 510–517. ISBN 978-1-4673-1226-4. Disponível em: <http://dblp.uni-trier.de/db/conf/cvpr/cvpr2012.html#AlahiOV12>. Citado na página 26.

ALCANTARILLA, P. F.; BARTOLI, A.; DAVISON, A. J. KAZE features. In: *Eur. Conf. on Computer Vision (ECCV)*. [S.l.: s.n.], 2012. Citado na página 26.

ALCANTARILLA, P. F.; NUEVO, J.; BARTOLI, A. Fast explicit diffusion for accelerated features in nonlinear scale spaces. In: *British Machine Vision Conf. (BMVC)*. [S.l.: s.n.], 2013. Citado na página 26.

ALSINA, P. J. et al. Navegação e controle de robôs móveis. *XIV Congresso Brasileiro de Automática*, 2002. Citado na página 2.

AROCA, R. et al. Increasing students' interest with low-cost cellbots. *Education, IEEE Transactions on*, v. 56, n. 1, p. 3–8, Feb 2013. ISSN 0018-9359. Citado na página 7.

BAY, H. et al. Speeded-up robust features (surf). *Comput. Vis. Image Underst.*, Elsevier Science Inc., New York, NY, USA, v. 110, n. 3, p. 346–359, jun. 2008. ISSN 1077-3142. Disponível em: <http://dx.doi.org/10.1016/j.cviu.2007.09.014>. Citado na página 26.

BISWAS, J.; VELOSO, M. M. Wifi localization and navigation for autonomous indoor mobile robots. In: *IEEE International Conference on Robotics and Automation, ICRA 2010, Anchorage, Alaska, USA, 3-7 May 2010*. IEEE, 2010. p. 4379–4384. Disponível em: <http://dx.doi.org/10.1109/ROBOT.2010.5509842>. Citado 4 vezes nas páginas 11, 19, 20 e 27.

BORENSTEIN, J.; FENG, L. Measurement and correction of systematic odometry errors in mobile robots. *Robotics and Automation, IEEE Transactions on*, v. 12, n. 6, p. 869–880, Dec 1996. ISSN 1042-296X. Citado na página 3.

"BULLET". "2016". Disponível em: <"http://bulletphysics.org">. Citado na página 29.

CALONDER, M. et al. Brief: binary robust independent elementary features. In: *Proceedings of the 11th European conference on Computer vision: Part IV*. Berlin, Heidelberg: Springer-Verlag, 2010. (ECCV'10), p. 778–792. ISBN 3-642-15560-X, 978-3-642-15560-4. Disponível em: <http://dl.acm.org/citation.cfm?id=1888089.1888148>. Citado na página 26.

CANEDO-RODRIGUEZ, A. et al. Mobile robot positioning with 433-mhz wireless motes with varying transmission powers and a particle filter. *Sensors*, v. 15, n. 5, p. 10194, 2015. ISSN 1424-8220. Disponível em: <http://www.mdpi.com/1424-8220/15/5/10194>. Citado na página 30.

CRAIG, J. *Introduction to Robotics: Mechanics and Control*. Pearson/Prentice Hall, 2005. (Addison-Wesley series in electrical and computer engineering: control engineering). ISBN 9780201543612. Disponível em: <https://books.google.com.br/books?id=MqMeAQAAIAAJ>. Citado na página 1.

DOUCET, A. et al. Rao-blackwellised particle filtering for dynamic bayesian networks. In: *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2000. (UAI '00), p. 176–183. ISBN 1-55860-709-9. Disponível em: <http://dl.acm.org/citation.cfm?id=647234.720075>. Citado na página 20.

GEIGER, A. et al. Vision meets robotics: The kitti dataset. *International Journal of Robotics Research (IJRR)*, 2013. Citado na página 27.

GOOGLE. *Página principal do Android*. 2015. Página principal do Android. Disponível em: <http://www.android.com/>. Acesso em: 25 jul. 2015. Citado na página 22.

HARRIS, C.; STEPHENS, M. A combined corner and edge detector. In: *In Proc. of Fourth Alvey Vision Conference*. [S.l.: s.n.], 1988. p. 147–151. Citado na página 26.

HARTLEY, R. I.; ZISSERMAN, A. *Multiple View Geometry in Computer Vision*. Second. [S.l.]: Cambridge University Press, ISBN: 0521540518, 2004. Citado na página 25.

ITSEEZ. *Página principal da biblioteca OpenCV*. 2015. Página principal da biblioteca OpenCV. Disponível em: <http://opencv.org/>. Acesso em: 25 jul. 2015. Citado na página 23.

LATOMBE, J.-C. *Robot Motion Planning*. Norwell, MA, USA: Kluwer Academic Publishers, 1991. ISBN 079239206X. Citado na página 12.

LEUTENEGGER, S.; CHLI, M.; SIEGWART, R. Brisk: Binary robust invariant scalable keypoints. In: *Computer Vision (ICCV), 2011 IEEE International Conference on*. [S.l.: s.n.], 2011. p. 2548–2555. ISSN 1550-5499. Citado na página 26.

LIU, H. et al. Survey of wireless indoor positioning techniques and systems. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, v. 37, n. 6, p. 1067–1080, Nov 2007. ISSN 1094-6977. Citado na página 18.

LOWE, D. Object recognition from local scale-invariant features. In: *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*. [S.l.: s.n.], 1999. v. 2, p. 1150–1157 vol.2. Citado na página 26.

LYE, Z. et al. Localization and feature recognition: Implementation on an indoor navigator robot. In: *Mecatronics (MECATRONICS), 2014 10th France-Japan/ 8th Europe-Asia Congress on*. [S.l.: s.n.], 2014. p. 238–243. Citado na página 30.

MAIR, E. et al. Adaptive and generic corner detection based on the accelerated segment test. In: _____. *Computer Vision – ECCV 2010: 11th European Conference on Computer Vision, Heraklion, Crete, Greece, September 5-11, 2010, Proceedings, Part II*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010. p. 183–196. ISBN 978-3-642-15552-9. Citado na página 26.

MATAS, J. et al. Robust wide baseline stereo from maximally stable extremal regions. In: *Proceedings of the British Machine Vision Conference*. [S.l.]: BMVA Press, 2002. p. 36.1–36.10. ISBN 1-901725-19-7. Doi:10.5244/C.16.36. Citado na página 26.

"NBR9050". "2015". Disponível em: <"http://www.pessoacomdeficiencia.gov.br/app/sites/default/files/arquivos/%5Bfield_generico_imagens-filefield-description%5D_164.pdf">. Citado 2 vezes nas páginas 12 e 54.

"NEWTON Dynamics". "2016". Disponível em: <"http://newtondynamics.com">. Citado na página 29.

NISTÉR, D.; NARODITSKY, O.; BERGEN, J. R. Visual odometry. In: *CVPR (1)*. [s.n.], 2004. p. 652–659. Disponível em: <http://doi.ieeecomputersociety.org/10.1109/CVPR.2004.265>. Citado na página 14.

"OPEN Dynamic Engine (ODE)". "2016". Disponível em: <"http://www.ode.org/">. Citado na página 29.

OPENCV. *Common Interfaces of Descriptor Matchers — OpenCV 2.4.11.0 documentation*. Disponível em: <http://docs.opencv.org/modules/features2d/doc/common_interfaces_of_descriptor_matchers.html#bfmatcher-bfmatcher>. Citado na página 26.

OPENCV. *Common Interfaces of Feature Detectors — OpenCV 2.4.11.0 documentation*. Disponível em: <http://docs.opencv.org/modules/features2d/doc/common_interfaces_of_feature_detectors.html#densefeaturedetector>. Citado na página 26.

OPENCV. *Common Interfaces of Feature Detectors — OpenCV 2.4.11.0 documentation*. Disponível em: <http://docs.opencv.org/modules/features2d/doc/common_interfaces_of_feature_detectors.html#simpleblobdetector>. Citado na página 26.

ROHMER S. P. N. SINGH, M. F. E. "v-rep: a versatile and scalable robot simulation framework". In: *"Proc. of The International Conference on Intelligent Robots and Systems (IROS)"*. [S.l.: s.n.], "2013". Citado na página 28.

ROSTEN, E.; REITMAYR, G.; DRUMMOND, T. Real-time video annotations for augmented reality. In: *INTERNATIONAL SYMPOSIUM ON VISUAL COMPUTING*. [S.l.: s.n.], 2005. p. 294–302. Citado na página 26.

RUBLEE, E. et al. Orb: An efficient alternative to sift or surf. In: *Computer Vision (ICCV), 2011 IEEE International Conference on*. [S.l.: s.n.], 2011. p. 2564–2571. ISSN 1550-5499. Citado na página 26.

SANTANA, A. M. *Localização e Planejamento de Caminhos para um Robô Humanóide e um Robô Escravo com Rodas*. Dissertação (Mestrado) — Universidade Federal do Rio Grande do Norte, Brasília, jul. 2007. Citado na página 2.

SANTOS, A.; TARRATACA, L.; CARDOSO, J. The feasibility of navigation algorithms on smartphones using j2me. *Mobile Networks and Applications*, Springer US, v. 15, n. 6, p. 819–830, 2010. ISSN 1383-469X. Disponível em: <http://dx.doi.org/10.1007/s11036-010-0236-8>. Citado na página 8.

SCARAMUZZA, D.; FRAUNDORFER, F. Visual odometry [tutorial]. *IEEE Robot. Automat. Mag.*, v. 18, n. 4, p. 80–92, 2011. Disponível em: <http://dx.doi.org/10.1109/MRA.2011.943233>. Citado 3 vezes nas páginas 15, 16 e 25.

SHI, J.; TOMASI, C. *Good Features to Track*. Ithaca, NY, USA, 1993. Citado na página 26.

SLEESWYK, A. W. Vitruvius' waywiser. In: . [S.l.]: Archives Internationales D'Histoire des Sciences, 1979. v. 29, p. 11–22. Citado na página 11.

THRUN, S.; BURGARD, W.; FOX, D. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. [S.l.]: The MIT Press, 2005. ISBN 0262201623. Citado 7 vezes nas páginas 11, 4, 5, 12, 20, 22 e 28.

"VORTEX". "2016". Disponível em: <"http://www.cm-labs.com/">. Citado na página 29.

Apêndices

APÊNDICE A – Tempos de execução no Computador

Tabela 24 – Tabela com as médias e desvios padrão em milissegundos dos tempos de execução dos algoritmos no Computador, utilizando a correspondência por *BruteForce*

	AKAZE	*BRISK*	*ORB*
AGAST	-	2503.041 ± 752.925	2501.477 ± 751.199
AKAZE	713.794 ± 334.721	513.591 ± 297.612	513.142 ± 298.652
BRISK	-	1580.642 ± 428.409	1586.186 ± 431.698
FAST	-	2422.412 ± 675.461	2419.171 ± 677.654
GFTT	-	476.197 ± 125.711	475.038 ± 124.885
HARRIS	-	416.950 ± 146.879	416.449 ± 146.318
KAZE	-	938.034 ± 287.820	940.273 ± 288.862
MSER	-	354.388 ± 136.627	354.797 ± 137.138
ORB	-	**301.527 ± 135.123**	301.977 ± 135.006

Tabela 25 – Tabela com as médias e desvios padrão em milissegundos dos tempos de execução dos algoritmos no Computador, utilizando a correspondência por *BruteForce-Hamming*

	AKAZE	*BRISK*	*ORB*
AGAST	-	1977.260 ± 898.316	1973.345 ± 896.065
AKAZE	575.588 ± 306.332	307.098 ± 124.454	304.149 ± 119.155
BRISK	-	1490.062 ± 517.680	1483.084 ± 514.518
FAST	-	1643.245 ± 820.669	1647.166 ± 825.746
GFTT	-	268.045 ± 125.017	265.957 ± 120.396
HARRIS	-	241.088 ± 121.216	241.090 ± 120.946
KAZE	-	658.160 ± 134.588	657.765 ± 134.520
MSER	-	326.874 ± 105.252	328.292 ± 106.944
ORB	-	**144.315 ± 89.589**	145.412 ± 89.313

Tabela 26 – Tabela com as médias e desvios padrão em milissegundos dos tempos de execução dos algoritmos no Computador, utilizando a correspondência por *BruteForce-Hamming Look-up Table*

	AKAZE	*BRISK*	*ORB*
AGAST	-	1975.501 ± 895.998	1971.790 ± 895.842
AKAZE	575.303 ± 306.069	307.061 ± 123.525	305.014 ± 119.546
BRISK	-	1488.698 ± 518.390	1482.935 ± 514.977
FAST	-	1634.550 ± 814.657	1637.885 ± 816.101
GFTT	-	265.074 ± 119.516	267.100 ± 120.089
HARRIS	-	241.251 ± 121.176	240.864 ± 120.701
KAZE	-	658.103 ± 137.738	658.231 ± 134.351
MSER	-	328.445 ± 106.913	326.736 ± 104.779
ORB	-	**144.235 ± 89.513**	144.643 ± 88.798

APÊNDICE B – Trajetórias do robô com relação a trajetória real

B.1 Quanto a variação da quantidade de *features*

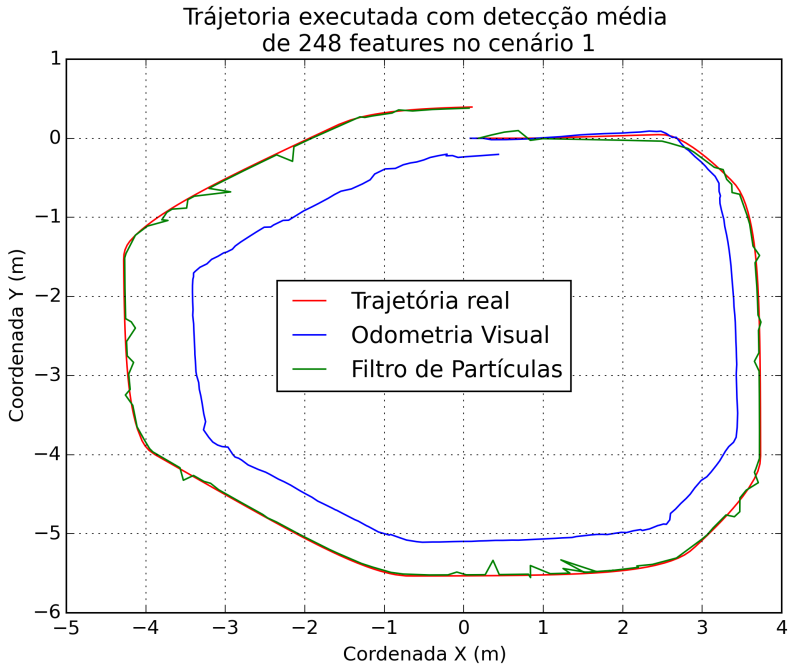


Figura 48 – Comparação da trajetória do robô utilizando uma média de 248 *features*.

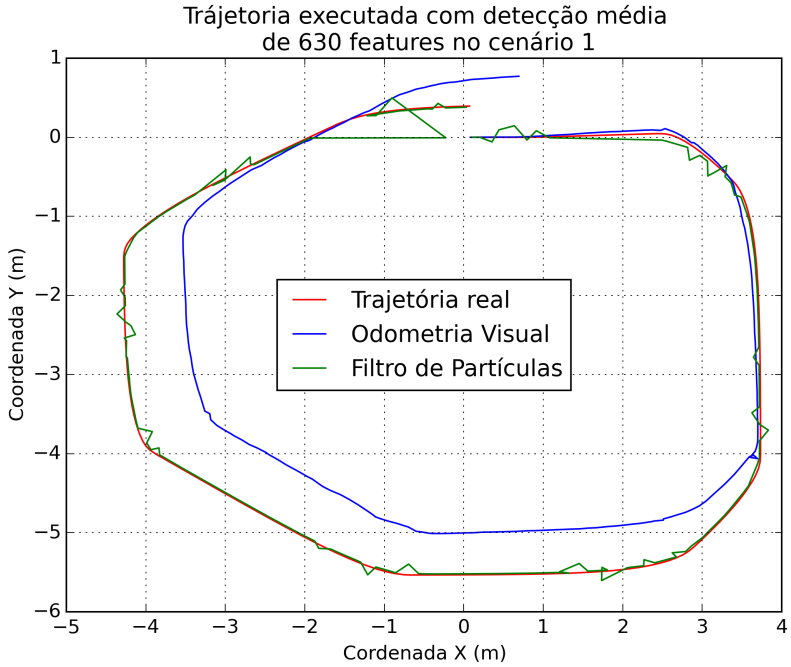


Figura 49 – Comparação da trajetória do robô utilizando uma média de 630 *features*.

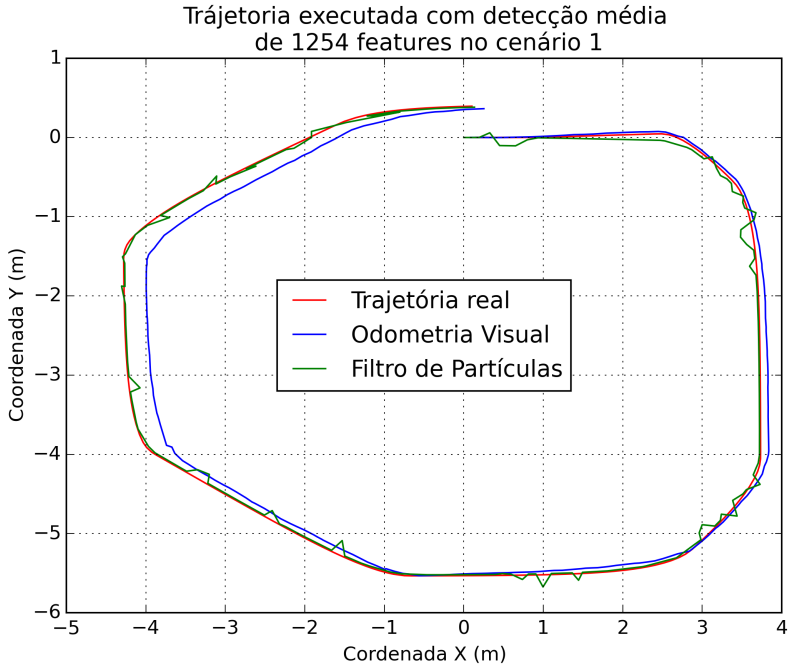


Figura 50 – Comparação da trajetória do robô utilizando uma média de 1254 *features*.

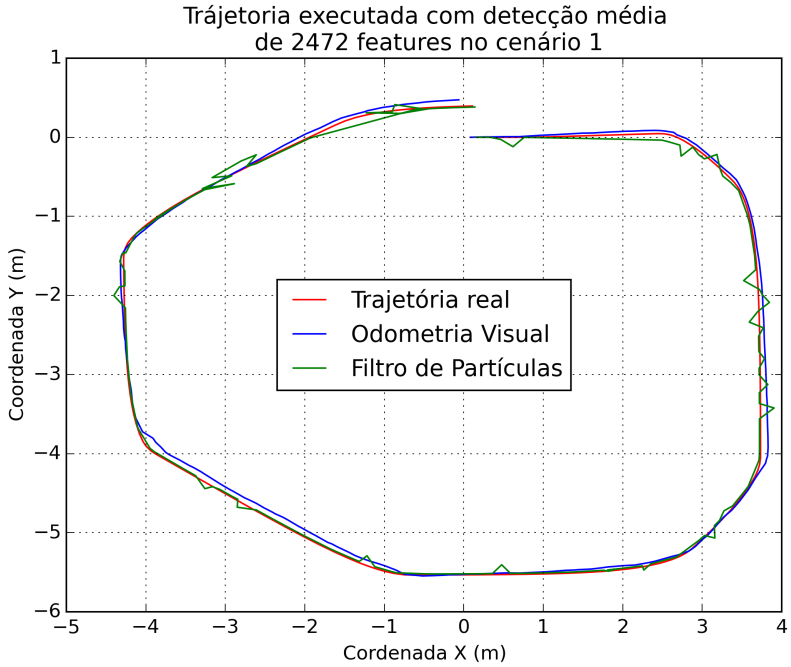


Figura 51 – Comparação da trajetória do robô utilizando uma média de 2472 *features*.

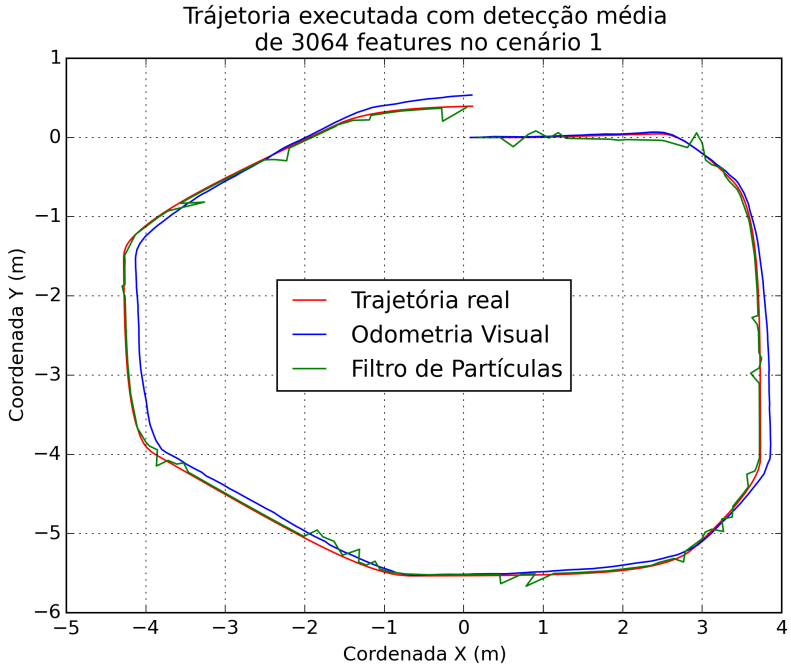


Figura 52 – Comparação da trajetória do robô utilizando uma média de 3064 *features*.

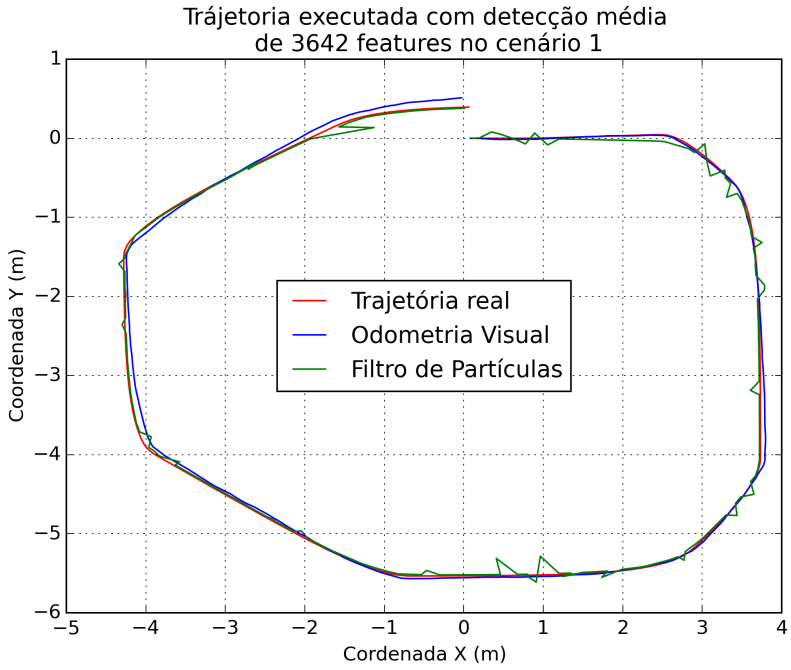


Figura 53 – Comparação da trajetória do robô utilizando uma média de 3642 *features*.

B.2 Quanto a variação da quantidade de partículas

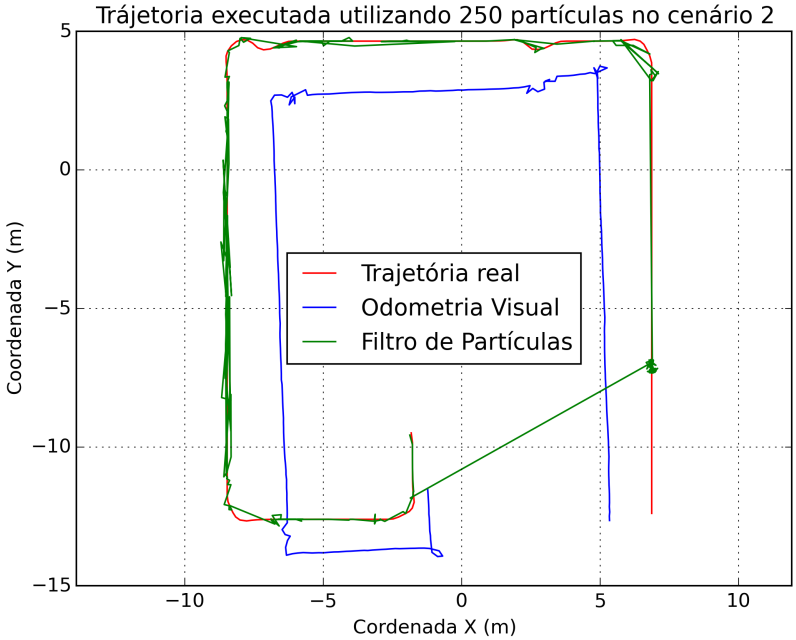


Figura 54 – Comparação da trajetória do robô utilizando 250 partículas.

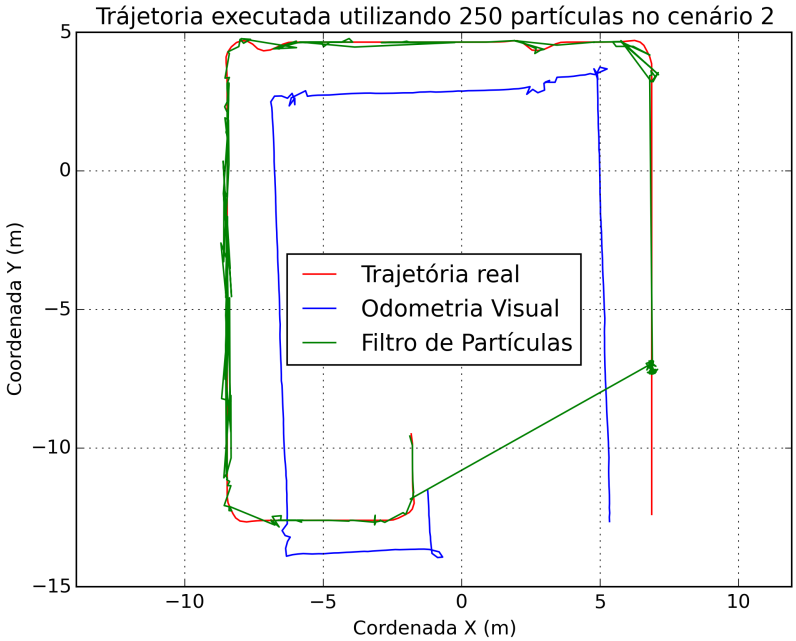


Figura 55 – Comparação da trajetória do robô utilizando 500 partículas.

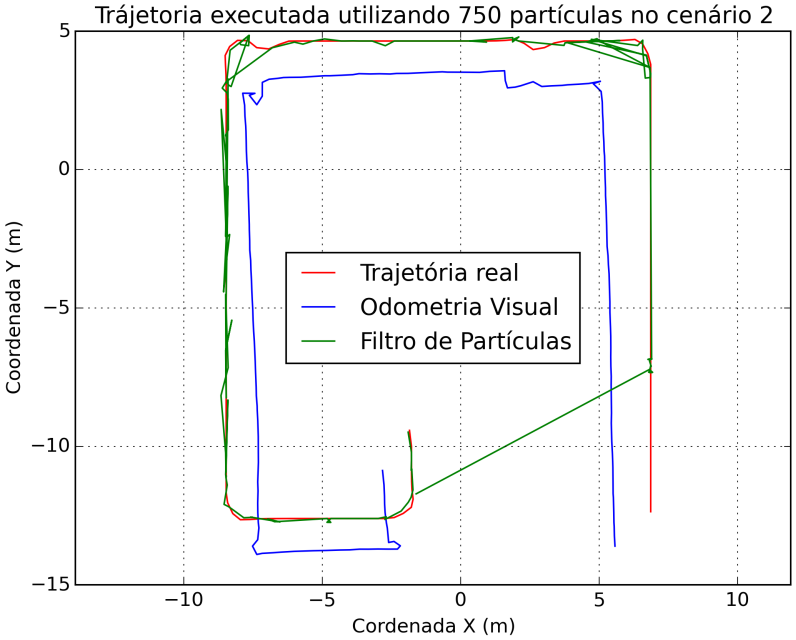


Figura 56 – Comparação da trajetória do robô utilizando 750 partículas.

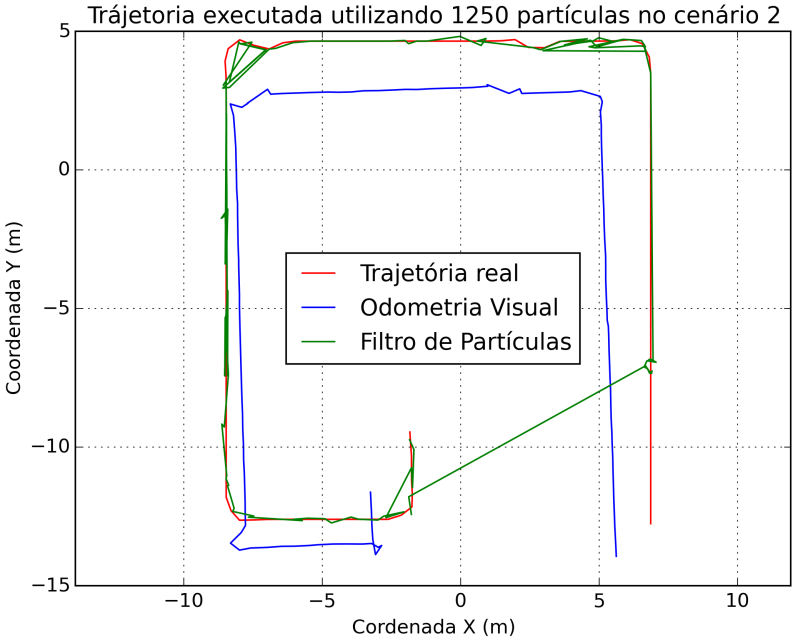


Figura 57 – Comparação da trajetória do robô utilizando 1250 partículas.

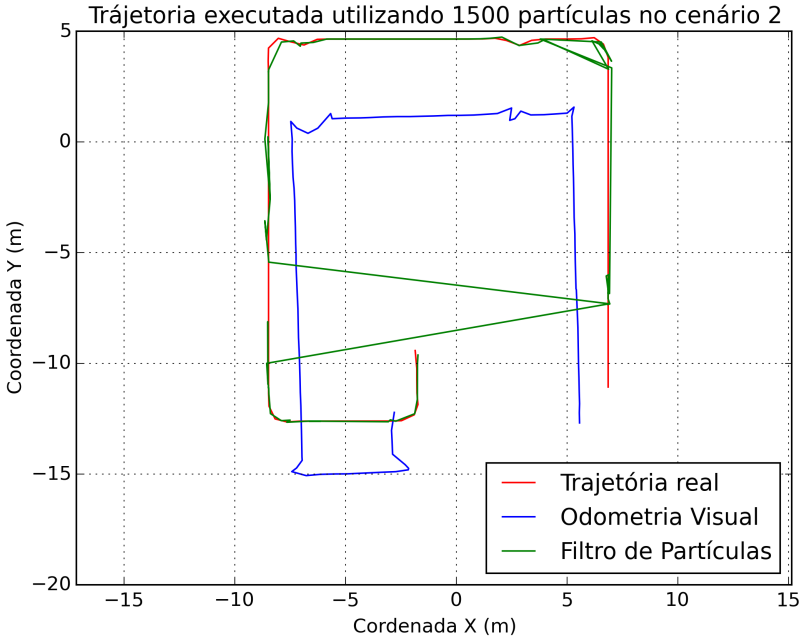


Figura 58 – Comparação da trajetória do robô utilizando 1500 partículas.

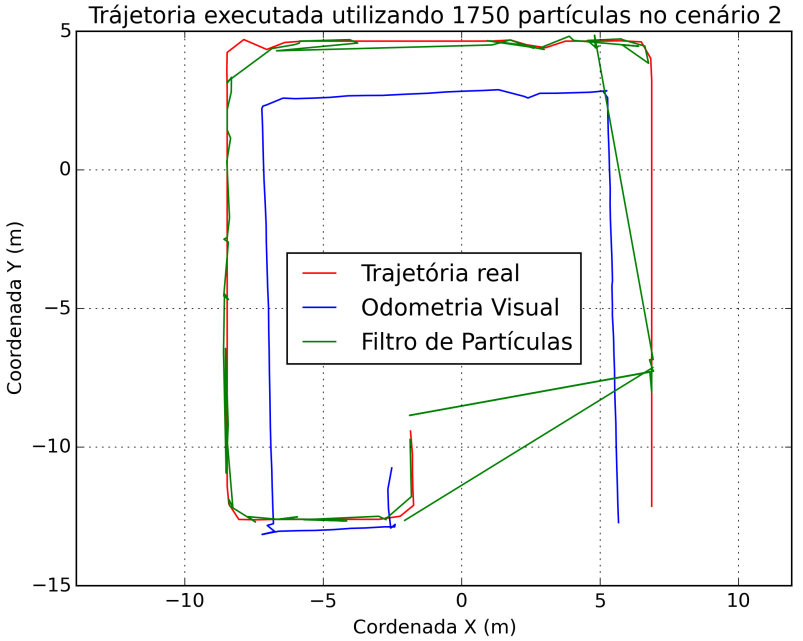


Figura 59 – Comparação da trajetória do robô utilizando 1750 partículas.

B.3 Quanto a variação da quantidade de *APs* no ambiente

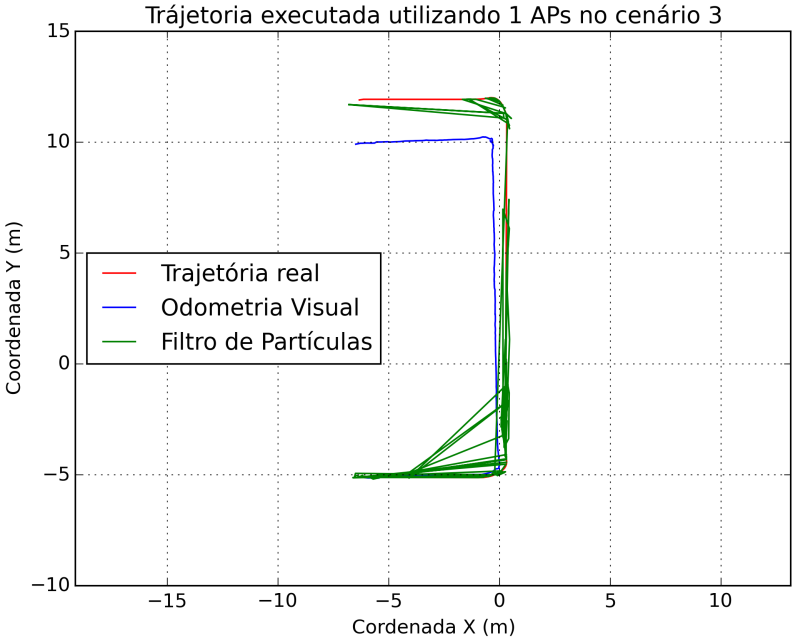


Figura 60 – Comparação da trajetória do robô utilizando 1 *Access Point*.

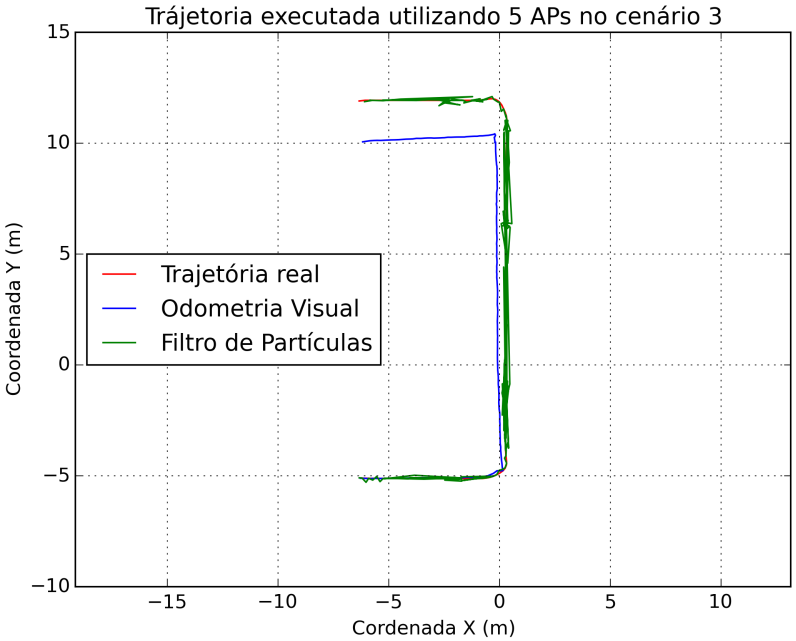


Figura 61 – Comparação da trajetória do robô utilizando 5 *Access Points*.

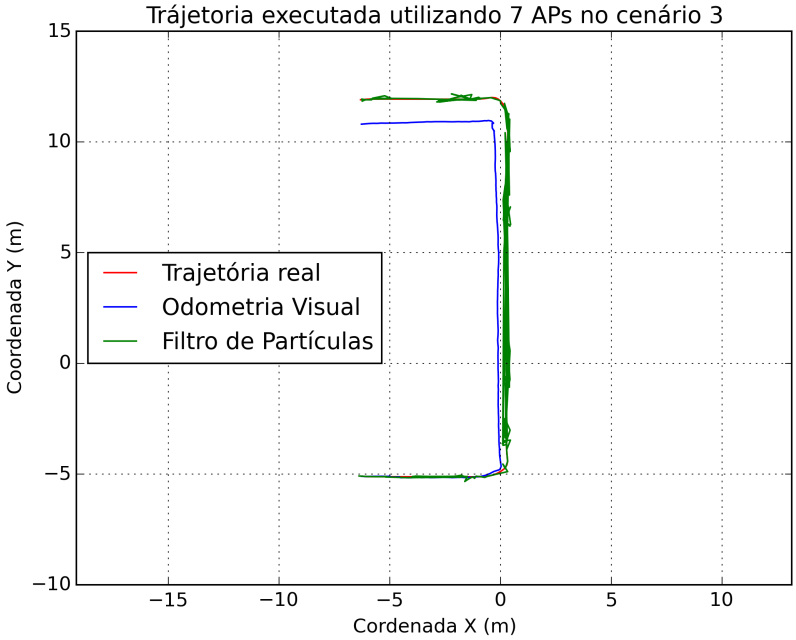


Figura 62 – Comparação da trajetória do robô utilizando 7 *Access Points*.

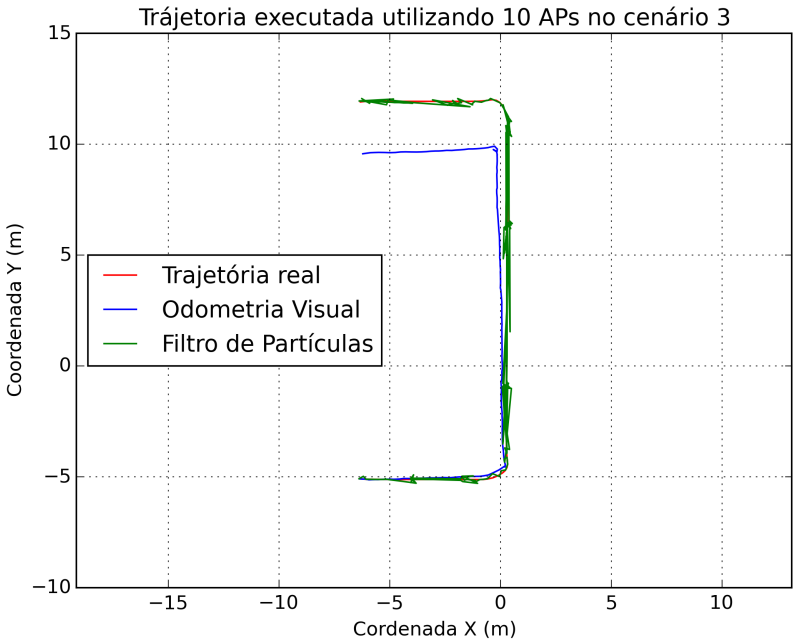


Figura 63 – Comparação da trajetória do robô utilizando 10 *Access Points*.