FILIPE LEUCH BONFIM
MICHAEL LIANG

APLICAÇÕES ESCALÁVEIS COM *MEAN STACK*

Trabalho de Graduação apresentado à disciplina de CI083 - Trabalho de Graduação em Arquitetura e Organização de Computadores II como requisito à conclusão do curso de Bacharelado em Ciência da Computação, Setor de Ciências Exatas Universidade Federal do Paraná.
Orientador: Bruno Mller Junior.

CURITIBA

2014

SUMÁRIO

LISTA DE FIGURAS **iii**

RESUMO **iv**

ABSTRACT **v**

1 INTRODUÇÃO **1**

1.1 Objetivo 3

1.2 Organização do trabalho 3

2 TÉCNOLOGIAS E REVISÃO BIBLIOGRÁFICA **5**

2.1 Localização 5

2.1.1 Localização de Markov 6

2.1.2 Localização filtro de Kalman 7

2.2 Construção de Mapas 7

2.2.1 Mapas Métricos 8

2.2.2 Mapas Topológicos 8

2.3 Problema de auto localização e mapeamento simultâneos 9

3 CONCEITUÇÃO E IDÉIA GERAL **11**

3.1 TOA(*Time of Arrival*) 11

3.2 TDOA(*Time Difference of Arrival*) 12

3.3 AOA(*Angle of Arrival*) 13

3.4 RSS(*Received Signal Strength*) 14

4 IMPLEMENTAÇÃO **17**

4.1 MainActivity 18

4.2 WifiInterface 20

4.3 Map 20

4.4 Algoritmo de Localização baseado em RSS *fingerprinting* 20
4.5 Análise e Resultados Obtidos 21
4.5.1 Testes realizados 22
4.5.2 Análise dos Resultados Obtidos 24

5 CONCLUSÃO **25**
5.1 Trabalhos Futuros 25

BIBLIOGRAFIA **30**

6 APÊNDICE A **31**
6.1 Arquitetura 32
6.1.1 Aplicações 32
6.1.2 *Framework* de Aplicações 32
6.1.3 Bibliotecas 33
6.1.4 *Android Runtime* 34
6.1.5 Linux Kernel 35
6.2 Componentes de uma Aplicação 35
6.2.1 Activity 36
6.2.2 Services 38
6.2.3 Broadcast Receiver 38
6.2.4 Content Providers 39

LISTA DE FIGURAS

2.1 Regra de Bayes. 6

3.1 Anglel of Arrival[26]. 13

3.2 Modelo Log-normal [37]. 14

3.3 Influência de um obstáculo na localização de um nodo[25]. 15

4.1 Esquema de interação entra as classes 18

4.2 Tela inicial, tela na obtenção do fingerprint e tela da obtenção da posição corrente do aparelho. 19

4.3 Calculo de um centroide de um conjunto de pontos. 21

4.4 Mapa do primeiro andar do Departamento de Informática da UFPR. . . . 22

6.1 Arquitetura Android[27]. 32

6.2 Ciclo de vida de uma *activity*[27]. 37

RESUMO

Nos últimos anos, tem-se observado um grande crescimento na utilização de sistemas robóticos no dia a dia. Cada vez mais pesquisadores em robótica têm se concentrado no desenvolvimento de robôs móveis, fazendo com que os robôs possam se mover e interagir com o ambiente de forma autônoma, o que abre um vasto campo de novas aplicações e, consequentemente, muitos desafios, como: localização, construção de mapas e o problema de auto localização e construção de mapas de ambiente simultâneos(*Simultaneous Localization and Map Building - SLAM*).

Este trabalho propõe um sistema de localização para robôs móveis, baseado no método empírico sugerido no artigo [4], que provê a localização de um terminal móvel em um ambiente *indoor*. O método é baseado na força do sinal recebido(*Received Signal Strength* - RSS), que é muito explorada em em técnicas de localização em uma rede de sensores sem fio pela fácil aplicabilidade. Serão apresentadas também, outras técnicas utilizadas na localização em uma rede de sensores sem fio.

Todo o sistema foi desenvolvido na plataforma Android, pela sua fácil incorporação à robótica devido ao grande quantidade de sensores suportados. Serão brevemente comentadas algumas características deste sistema operacional e de uma aplicação Android.

Os resultados obtidos pelo sistema proposto mostram que:

- 22,8% das estimativas possuem precisão de 1 metro ou menos.
- 77,1% das estimativas possuem precisão de 3 metros ou menos.
- 91,4 % das estimativas possuem precisão de 5 metros ou menos.

Palavras-chave: RSS, Localização, RSS fingerprint, Robós Móveis, Android, WSNs.

ABSTRACT

In the last years, we have watched a big grow in the daily use of robotic systems. More and more researchers in robotics have focus on the mobile robots development, they makes that robots be capable of move and interact with the environment, what generates a huge field of new applications and new challenges like: localization, map building and the problem of simultaneous localization and map building(SLAM).

This work proposes a localization system for mobile robots based on the article [4], which provides a localization method of a mobile terminal in a indoor environment. The method is based on the received signal strength(RSS), which is very used in localization techniques in the wireless sensors networks due the easy applicability. Other techniques for localization in the wireless sensors networks are shown in this work.

The entire system was built on the Android plataform by the easy incorporation on robotics due the big amount of sensors supported. Some features of a Android application and of this operation system are brief commented.

The results got by the proposed system shown that:

- 22,8% of the estimatives have precision of 1 meter or less.
- 77,1% of the estimatives have precision of 3 meter or less.
- 91,4 % of the estimatives have precision of 5 meter or less.

Keywords: RSS, Localization, RSS fingerprint, Mobile Robots, Android, WSNs.

CAPÍTULO 1

INTRODUÇÃO

Robôs móveis são sistemas incorporados no mundo real que se movem autonomamente e interagem com ele para realizar suas tarefas[33]. A utilização de tais robôs já é frequente hoje em dia e as tarefas a eles designadas estão aumentando em complexidade. Eles podem ser utilizados em várias aplicações, como limpeza, corte de grama, detectar riscos, explorações de ambientes desconhecidos, vigilância autônoma, e assistência a idosos ou pessoas com alguma incapacidade. Robôs podem cooperar pra realizar tarefas em comum, como encontrar e resgatar sobreviventes de um terremoto em território urbano [21].

Para o robô se mover de forma autônoma ele deve ser capaz de realizar tarefas como a de planejamento de caminho, navegação, localização, evitar obstáculos, controle de motores, construção e atualização de mapas.

Na navegação do robô é essencial que o robô conheça o ambiente na qual ele realizará sua tarefa. Um mapa é uma representação espacial utilizada para registrar a localização de elementos relevantes [33]. Tipicamente, existem duas abordagens para a representação de mapas de ambiente [6]: mapas métricos e mapas topológicos. Mapas métricos contêm informação da geometria do ambiente, da posição dos objetos e distâncias entre esses. Os mapas topológicos não possuem qualquer informação sobre a geometria do ambiente, à eles são representados por elos conectados a nós [6].

Na navegação com mapas topológicos, utilizam os nós do mapa para que o robô possa tomar certas ações, como por exemplo, virar à direita ou à esquerda, e reconhecer marcos para se localizar no ambiente. O reconhecimento dos marcos visuais dá ao robô uma localização qualitativa no ambiente, obtendo sua posição em termos do quanto está mais próximo ou mais distante do alvo. Na navegação com mapas geométricos a localização é quantitativa, sabendo o robô a sua posição exata no ambiente [6].

Na navegação autônoma, os elementos representados em um mapa podem ser utilizados

para diversas finalidades. Por exemplo, eles podem ser usados para planejar um caminho entre a posição atual do robô e seu destino [19], especialmente se o mapa representar as áreas por onde ele for permitido navegar. Os mapas podem também ser utilizados para localizar o robô: comparando os elementos sensoreados no ambiente com aqueles registrados no mapa, o robô pode inferir o lugar ou possíveis lugares onde está.

Há um conjunto de algoritmos de localização que utilizam uma rede de sensores sem fio (*Wireless sensor networks* - WSNs), na localização do robô, onde o cálculo da localização depende das informações de localização dos nodos (podem ser estáticos ou móveis) da rede[36]. A posição de um certo nodo, pode ser obtida, a partir da posição ou direção dos nodos conhecidos por ele.

Tempo de chegada (*Time of Arrival* - TOA) [16], ângulo de chegada (*Angle of Arrival* - AOA) [24], diferença de chegada de sinais diferentes (*Time Difference of Arrival* - TDOA) [32] e força do sinal recebido (*Received Signal Strength* - RSS) [4], são alguns das abordagens feitas por algoritmos que utilizam WSNs, para calculo da localização.

Este trabalho traz uma proposta de localização *indoor* baseado em RSS para robôs móveis, utilizando como base o método empírico sugerido no artigo[4]. Nesse trabalho é utilizada a plataforma Android, pois ela [3] é uma plataforma de fácil desenvolvimento, com documentação farta [1], e possui *drivers* para câmera, wifi, acelerômetro, *bluetooth*, microfone, compasso e GPS [23].

1.1 Objetivo

Implementar um sistema de localização *indoor* baseado em RSS para robôs móveis, desenvolvendo o trabalho de graduação realizado por Alexandre Umezaki e Walter Mazuroski em 2010, no qual implementaram o método empírico proposto por Bahl e Padmanabhan no artigo [4].

O sistema implementado por Alexandre Umezaki e Walter Mazuroski é dividido em duas etapas. Na primeira etapa há a coleta de *"fingerprints"* de RSS dos *access points*, onde são armazenadas em uma tabela.

A segunda etapa, na qual é realizado o processo de localização em si, o sistema recebe a força do sinal wifi de três APs, e calculava a distância euclidiana dos três APs observados em relação as amostras feitas na primeira etapa. A amostra com a menor distância euclidiana é dada como a provável localização do nodo. Esse sistema foi feito para Linux, utilizando a linguagem C e as APIs (*Application Programming Interface*) Wireless Tools e o Wireless Extension[35] para coletar as amostras de força do sinal.

O sistema desenvolvido nesse trabalho se diferencia do sistema previamente apresentado em especialmente dois aspectos. Primeiro a plataforma adotada: Android. Como mostrado anteriormente, o Android possui muitos recursos que podem ser empregados na robótica e a obtenção de informações sobre os *access points* são facilmente obtidos pelo *framework* de aplicativos dessa plataforma como pode ser visto em[34].

A segunda diferença está na obtenção do posição do nodo. O sistema aqui implementado, gera um conjunto de pontos um conjunto de pontos que definem uma região no qual o nodo possa estar. A localização desse nodo é estimada através do cálculo do centroide desse conjunto de pontos.

1.2 Organização do trabalho

Este trabalho está dividido em 5 partes. A primeira parte contempla as atividades de construção de mapas e localização que um robô móvel deve desempenhar. Ela traz também, um dos grandes desafios da robótica, o problema de auto localização e construção

de mapas de ambiente simultâneos(SLAM).

Localização usando rede de sensores sem fio é o tema da segundo parte, onde são mostradas as principais técnicas utilizadas nesse tópico.

A terceira parte aborda a plataforma Android, mostrando a sua arquitetura e as principais característica de uma aplicações Android.

Na quarta parte são mostradas a proposta de localização para robôs móveis utilizada nesse trabalho, como foi implementado e os resultados obtidos.

A ultima parte traz a conclusão desse trabalho e os possíveis trabalhos futuros.

CAPÍTULO 2

TÉCNOLOGIAS E REVISÃO BIBLIOGRÁFICA

Neste capitulo são discutidas as principais tarefas que o robô deve desempenhar para que ele possa concluir seu objetivos de navegação de forma autônoma e da melhor maneira possível, sendo elas: localização e construção de mapas.

2.1 Localização

A obtenção da posição e orientação (pose) do robô pode ser feita através da identificação e subsequente triangulação por ângulos e por distância dos *landmarks* percebidos pelo robô e previamente conhecidos, onde a identificação dos *landmarks* é feita fazendo observações do ambiente utilizando seus sensores(sonar, laser, câmera).

No cálculo da localização, o robô deve ser capaz de lidar com os erros de medição dos sensores, incertezas (que tendem a crescer com o deslocamento do robô) e informações incompletas [28]. Portanto, ao invés de calcular a posição exata, o que pode ser feito é calcular a probabilidade do robô estar numa certa posição. Daí a necessidade da localização probabilística, na qual, a incerteza é representada utilizando teoria da probabilidade: ao invés de dar a melhor estimativa da configuração atual do robô, a localização probabilística dá a distribuição de probabilidade de todas as possíveis configurações do robô [28]. Essa distribuição de probabilidade é chamada *belief*.

Quando o robô se movimenta a incerteza de sua posição aumenta. Fazendo observações do ambiente e mesclando os dados obtidos com a estimação da odometria, o robô pode combinar essas informações com o *belief* anterior ao deslocamento. Deste modo, a cada movimento o robô pode obter uma melhor estimativa de sua real posição, isso é chamado modelo de movimento e percepção.

A cada deslocamento do robô é necessário fazer a atualização da distribuição de probabilidade de sua configuração. A atualização pode ser dividida em 2 passos [28]:

- Atualização de ação: o robô se move e estima sua posição através de seus sensores nesse passo a incerteza aumenta.

- Atualização de percepção: o robô faz uma observação usando seus sensores e corrige sua posição, combinando seu *belief* com a probabilidade de fazer essa observação. Aqui a incerteza diminui.

No cálculo da localização probabilística é necessário ter: a distribuição de probabilidade inicial, o modelo de erro estatístico dos sensores e o mapa do ambiente. Há duas principais abordagens para solução da localização probabilística de robôs móveis: localização de Markov e filtro de Kalman, descritos a seguir.

2.1.1 Localização de Markov

A localização de Markov usa um *grid* para representar a configuração do robô, onde cada célula do *grid* contém a probabilidade de o robô estar nela [28]. A distribuição de probabilidade associada à percepção dos sensores também é discretizada. Durante as etapas de ação e percepção todas as células do *grid* são atualizadas.

A atualização na etapa de ação é feita através da convolução da distribuição do *belief* inicial com a distribuição da probabilidade da possível pose do robô após seu deslocamento, com a incerteza aumentada, segundo o modelo de deslocamento. Na etapa da percepção, a atualização e feita fazendo a convolução do *belief* inicial com o modelo estatístico de erro dos sensores. A convolução pode ser feita através da regra de Bayes[28].

A ideia principal da regra de Bayes é que a probabilidade de um evento A dado um evento B depende não apenas do relacionamento entre os eventos A e B, mas também da probabilidade marginal (ou "probabilidade simples") da ocorrência de cada evento:

$$\Pr(A|B) = \frac{\Pr(B|A)\,\Pr(A)}{\Pr(B)}$$

Figura 2.1: Regra de Bayes.

Como todas as células são atualizadas nas etapas de ação e percepção, a localização de Markov requer grande quantidade de processamento e memória.

2.1.2 Localização filtro de Kalman

Na localização filtro de Kalman, assume-se que a distribuição de probabilidade da configuração do robô e o modelo dos sensores, são continuas e gaussianas [29]. Como a distribuição gaussiana é descrita através da média e variância, somente essas duas variáveis são atualizadas nas etapas de ação e percepção. Portanto, seu custo computacional é pequeno se comparado com a localização de Markov.

O filtro de Kalman produz estimativas dos valores reais de grandezas medidas e valores associados predizendo um valor, estimando a incerteza do valor predito e calculando uma média ponderada entre o valor predito e o valor medido. O peso maior é dado ao valor de menor incerteza. As estimativas geradas pelo método tendem a estar mais próximas dos valores reais que as medidas originais pois a média ponderada apresenta uma melhor estimativa de incerteza que ambos os valores utilizados no seu cálculo.

O filtro de Kalman combina uma predição da posição atual do robô a com uma nova medida usando uma média ponderada. A ideia dos pesos é que valores com menor incerteza estimada sejam mais "confiáveis". Os pesos são calculados através da covariância, uma medida da incerteza estimada da predição do estado do sistema [31].

O resultado da média ponderada é uma nova estimativa do estado, que se localiza entre o estado predito e o estado medido, apresentando uma melhor incerteza estimada que qualquer um dos dois unicamente. Este processo é repetido a cada fase, com a nova estimativa e sua covariância gerando a predição usada na próxima iteração. Isto significa que o filtro de Kalman funciona recursivamente e requer apenas a última estimativa - não o histórico completo - do estado de um sistema para calcular o próximo estado[29].

2.2 Construção de Mapas

A tarefa de mapeamento corresponde à atribuição de valores aos elementos do mapa, relacionando cada um a uma certa posição nele [33]. O tipo ideal de mapa a ser utilizado ou construído por um robô móvel depende da tarefa e do ambiente onde este esteja inserido. Também depende das características do robô, tais como os tipos de sensores que

ele tem, bem como a forma com ele se move[33]. Há duas abordagens principais para a representação de mapas de ambiente [6]. São os mapas métricos e topológicos, que serão discutidos a seguir.

2.2.1 Mapas Métricos

No caso da construção de mapas métricos, o objetivo é obter um mapa detalhado do ambiente, com informações sobre a forma e o tamanho dos objetos e os limites das áreas livres para a navegação, tais como corredores, quartos, trilhas e estradas. Para representar essa complexa e detalhada informação, o mapa é geralmente dividido em uma densa rede de forma que cada célula contenha informações sobre a ocupação desse espaço por um objeto e, possivelmente, outras características ambientais que estão sendo mapeadas [33].

Com à alta densidade de informação nesses mapas, o resultado da localização é mais preciso e menos sujeito a ambiguidades [33]. Como os mapas métricos demandam uma grande quantidade de informações, eles exigem muita capacidade de processamento e armazenamento.

Atualmente grande parte dos sistemas de mapeamento assume que o ambiente é estático durante o mapeamento. Se uma pessoa anda dentro do alcance dos sensores do robô durante o mapeamento, o mapa resultante conterá evidências a respeito de um objeto na localização correspondente. Além disso, se o robô retornar para esta localização e varrer a área uma segunda vez, sem a pessoa presente, a estimativa da posição será menos precisa, uma vez que as novas medições não contêm nenhum vestígio correspondente àquela pessoa. A exatidão reduzida do mapa resultante pode ter uma influência negativa no desempenho da localização robô.

2.2.2 Mapas Topológicos

No caso de mapas topológicos, o objetivo é construir uma estrutura relacional, normalmente um grafo, que de forma geral, registra informações sobre determinados elementos ou locais do ambiente, chamados marcos [33]. Assim, marcos e relações são os elementos dos mapas topológicos. Essas relações podem ser de vários tipos, tais como o desloca-

mento relativo entre dois marcos, a existência de um caminho entre eles, ou a quantidade de energia gasta em um caminho entre marcos. É fácil perceber que a informação contida no mapa topológico é menos detalhada e distribui-se de forma dispersa, concentrando-se apenas em pontos de interesse. Assim esta representação, é muito seletiva com relação à informação que ele registra [33].

O mapa topológico também demanda que o processamento dos dados dos sensores do robô, sejam feitos de forma mais abstrata, a fim de extrair informações sobre as localidades mais relevantes que devem ser consideradas como marcos.

2.3 Problema de auto localização e mapeamento simultâneos

O problema de auto localização e construção de mapas de ambiente simultâneos (*Simultaneous Localization and Map Building - SLAM*) consiste em um robô autônomo iniciar a navegação em uma localização desconhecida, em um ambiente desconhecido e então construir um mapa desse ambiente de maneira incremental, enquanto utiliza o mapa simultaneamente para calcular a sua localização [13].

O SLAM tem sido tema de várias pesquisas na área de robótica. A grande vantagem do SLAM é que elimina a necessidade um conhecimento a priori do ambiente. Há três abordagens principais que são utilizadas no problema do SLAM.

A primeira e mais popular deles usa o filtro de Kalman estendido para resolver o SLAM (Extended Kalman Filter SLAM - EKF SLAM) [30]. Ele fornece uma solução recursiva para o problema da navegação e uma maneira de calcular estimativas consistentes para a incerteza na localização do robô e nas posições dos marcos do ambiente, com base em modelos estatísticos para o movimento dos robôs e observações relativas dos marcos do ambiente[13].

O EKF SLAM resume toda a toda experiencia obtida pelo robô em um vetor de estados estendido Y, compreendendo a pose do robô e posição dos marcos do mapa, e a matriz de covariância P[30]. Quando o robô se desloca Y e P são atualizadas usando o EKF. Os marcos do ambiente são extraídos do ambiente de sua nova posição. Em seguida, o robô tenta associar esses marcos com os marcos previamente observados. A Reobservação dos

marcos é usada para atualizar a posição do robô. Os marcos que não foram observados previamente, são adicionados no mapa de marcos.

A segunda abordagem, chamada filtro de partículas SLAM, consiste em evitar a necessidade de estimativas absolutas da posição e de medições precisas das incertezas para utilizar conhecimento mais qualitativo da localização relativa dos marcos do ambiente e do robô para a construção do mapa global e planejamento da trajetória[6].

O filtro de partículas é um modelo matemático que representa a distribuição de probabilidade associada a um conjunto de partículas discretas. Uma partícula contém a estimativa da pose do robô com pesos associados(todos os pesos devem ser incrementados em 1) [30]. Por período de amostragem, cada partícula é modificada de acordo com o modelo do processo, incluindo a adição de ruído aleatório para simular o efeito do ruído nas variáveis de estado e, então, o peso de cada partícula é reavaliado com base na última informação sensorial.

As partículas com pesos próximos de zero são descartadas e recriam-se novas partículas com base naquelas que sobram. Quando o número efetivo de amostras está abaixo de um determinado limiar, geralmente calculado com base em uma percentagem das M partículas, então a população das M partículas é re-amostrada (resampling), eliminado-se probabilisticamente aquelas cujos pesos são pequenos e duplicando aquelas com pesos elevados [31].

O terceiro método é bastante amplo e afasta-se do rigor matemático do filtro de Kalman, ou do formalismo estatístico, retendo uma abordagem essencialmente numérica ou computacional para a navegação e para o problema de SLAM. Essa abordagem inclui o uso de correspondência com marcos artificiais do ambiente[6], ela é chamada de GraphSLAM. As posições do robô ao longo do tempo e os marcos correspondem a nós em um grafo [30]. As informações odométricas entre posições consecutivas e os marcos vistos em diferentes posições equivalem as arestas do grafo. O algoritmo é executado em 2 etapas. Na primeira etapa, o mesmo apenas acumula dados e constrói o grafo. Na segunda etapa, o o grafo é rearranjado para acomodar os dados obtidos. Diferente do EKF SLAM, o GraphSLAM estima a posição do robô durante todo o trajeto.

CAPÍTULO 3

CONCEITUÇÃO E IDÉIA GERAL

Este capitulo aborda as principais técnicas de localização utilizando WSNs, técnica para localização abordada neste trabalho.

WSN é uma rede de sensores que coleta informações em um campo monitorado. Este tipo de rede tem sido utilizada em várias aplicações, incluindo rastreamento de objetos, resgate, monitoramento de ambiente, entre outros [36]. Na maioria das WSNs os sensores são estáticos, mas a tendência em aplicações modernas é os sensores terem mobilidade.

Há um conjunto de algoritmos de localização que utilizam WSNs. Onde o cálculo da localização depende das informações de localização dos nodos (podem ser estáticos ou móveis) da rede. Esses algoritmos podem ser divididos em duas categorias: *range-based* e *range-free*. Algoritmos do tipo *range-free* [36] [38] geralmente requerem que os nodos conhecidos, estejam dentro do raio de comunicação dos sensores em comum. Eles tem sido muito explorados pois não precisam de *hardware* extra. Os algoritmos do tipo *range-based*, geralmente necessitam de algum tipo de *hardware* especial, onde a posição de um certo nodo pode ser obtida a partir da posição ou direção dos nodos conhecidos por ele.

3.1 TOA(*Time of Arrival*)

TOA é o tempo medido em que um sinal (rádio frequência, acústicos, ou outros) chega a um receptor pela primeira vez depois de emitido[16]. O valor medido é o tempo de transmissão somado ao atraso do tempo de propagação. Este atraso,$t_{i,j}$, entre a transmissão do sensor i e recepção do sensor j, é igual a distância entre o transmissor e o receptor, $d_{i,j}$, dividido pela velocidade de propagação do sinal, v_p . Esta velocidade para a rádio frequência é aproximadamente 10^6 vezes mais rápida que a velocidade do som [16], ou seja, 1 ms corresponde a 0,3 m na propagação sonora, enquanto para a rádio frequência, 1 ns corresponde a 0,3 m [16].

O ponto chave das técnicas baseadas em tempo é capacidade do receptor de estimar com precisão o tempo de chegada em sinais na linha da visão. Esta estimativa é prejudicada tanto pelo ruído aditivo quanto por sinais de multi percurso.

3.2 TDOA(*Time Difference of Arrival*)

Os algoritmos de TDOA fazem a medição da diferença no tempo de recepção de sinais de diferentes estações de base (Base Station's - BSs)), sem a necessidade de uma sincronização de todos os participantes BSs e terminais móveis(MT)) [32]. Na verdade, a incerteza entre os tempos de referência dos BSs e do MT pode ser removidas por meio de um cálculo diferencial. Por isso, apenas os BSs envolvidos no processo de estimativa de localização deve ser bem sincronizados.

Para um par de BSs, digamos i e j, o TDOA, T_{ij}, é dada por $T_{ij}TBSi - TBSj$, onde $TBSi$ e $TBSj$ são os tempos absolutos tomados pelo tempo de chegada nos BSs i e j, respectivamente. Supondo que o MT está em LOS (*line-of-sight*) com ambos os BSs, i e j, o MT deve situar-se em uma hipérbole. Uma segunda hipérbole, onde o MT deve estar, pode ser obtido através de uma medição adicional de TDOA envolvendo um terceiro BS. A posição do usuário pode ser identificada como o ponto de intersecção das duas hipérboles. A solução do sistema de equações pode ser encontrado com um método iterativo e minimização dos quadrados mínimos [32].

O método dos quadrados mínimos procura encontrar o melhor ajuste para um conjunto de dados tentando minimizar a soma dos quadrados das diferenças entre o valor estimado e os dados observados (tais diferenças são chamadas resíduos).

Queremos estimar valores de determinada variável y. Para isso, consideramos os valores de outra variável x que acreditamos ter poder de explicação sobre y conforme a fórmula:

$y = \alpha + \beta\ x + \varepsilon$

onde:

- α: Parâmetro do modelo chamado de constante (porque não depende de x).
- β: Parâmetro do modelo chamado de coeficiente da variável x.

- e: Erro - representa a variação de y que não é explicada pelo modelo.

Também temos uma base de dados com n valores observados de y e de x . O método dos mínimos quadrados ajuda a encontrar as estimativas de α e β.

O método dos mínimos quadrados minimiza a soma dos quadrado dos resíduos, ou seja, minimiza $\sum_{i=1}^{n} e_i^2$.

A ideia dessa técnica é que, minimizando a soma do quadrado dos resíduos, encontraremos a e b que trarão a menor diferença entre a previsão de y e o y realmente observado.

3.3 AOA(*Angle of Arrival*)

A Localização baseada em AOA envolve a medição do ângulo de chegada de um sinal de uma BS a um receptor ou vice-versa. Em qualquer um dos casos, uma única medida produz uma linha reta entre a BS e o receptor. A medida do ângulo de chegada com outra *base station* produzirá uma segunda linha reta e, a intersecção das duas linhas, vai fornecer a posição do dispositivo[26].

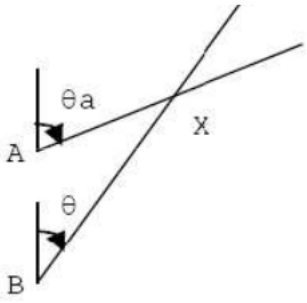

Figura 3.1: Anglel of Arrival[26].

Existem duas maneiras nas quais sensores medem o AOA [24]. O método mais comum é usar um vetor de sensores. Neste caso, cada nó sensor é composto de dois ou mais sensores individuais (microfones para sinais acústicos ou antenas de sinais de rádio frequência), cujas posições com relação ao centro do nó são conhecidos. A AOA é estimada a partir das diferenças de tempos de chegada de uma transmissão do sinal em cada um dos elementos do vetor de sensores.

A segunda abordagem para a estimativa do AOA, usa a razão RSS entre duas (ou mais) antenas direcionais localizadas no sensor. Duas antenas direcionais apontadas em direções diferentes, de tal maneira que suas hastes se sobrepõem, podem ser usadas para estimar a AOA partir da relação entre seus valores individuais RSS.

3.4 RSS(*Received Signal Strength*)

A localização baseada em RSS apresenta-se como uma das únicas a não necessitar de *hardware* extra e é uma técnica de fácil aplicação.

Há três métodos de localização que utilizam a medição de RSS em WSNs: estação base mais forte, modelo de propagação e *fingerprinting*. O primeiro método é o mais simples, a localização de um nodo da rede pode ser estimado como a posição do *Access Points*(AP) mais próximo, no qual o nodo se comunicou. Este método pode não atingir grande precisão devido a complexidade de WSNs em ambientes *indoor* e a limitação da cobertura do AP.

No método de modelo de propagação, a medição do sinal no receptor e o valor obtido indica a distância até o transmissor [25]. Sem nenhuma interferência, a distância de dois nodos i e j pode ser relacionada com a força do sinal medido, através do modelo log-normal apresentado em [37]:

$$\mathrm{rss(i,j)} = \mathrm{P_0} - 10nlog_{10}(\frac{d(i,j)}{d_0}) + X_\sigma$$

Figura 3.2: Modelo Log-normal [37].

Onde P_0 é a perda de percurso para a distância de referência, n á o expoente de perda de percurso, $d(i,j)$ é a distância entre os nodos i e j, d_0 é a distância de referencia, X_σ é uma variável aleatória Gaussiana de média zero com desvio padrão σ.

Tal forma de medição por muitas vezes é descartada, pois não leva em conta o ambiente onde está sendo aplicada a medição; logo, corresponde a um resultado não confiável, obstruções e obstáculos proporcionam erros de grande relevância, sendo esse um dos maiores problemas dos métodos de localização baseados em RSS, várias técnicas foram propostas para contornar esse problema como pode ser visto em [4][25][38][22] [37].

A figura 3.3 mostra como a estimativa da real posição de um nodo é afetado pelas interferências de obstrução:

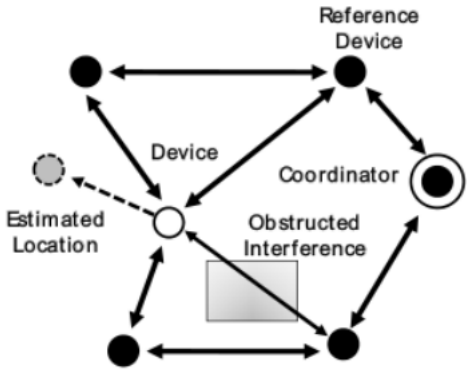

Figura 3.3: Influência de um obstáculo na localização de um nodo[25].

Recentemente, localização em WSNs baseado em RSS *fingerprinting* tem se tornado uma das técnicas mais exploradas em ambientes *indoor*[17][4][5]. Comparada com as outras duas técnicas anteriormente apresentadas, *fingerprinting* é facilmente empregada e é tolerante ao ruido do sinal, e isso permite uma maior precisão em relação às outras. Este método normalmente é composto por duas fases: aprendizado e determinação. Na primeira fase o objetivo é construir uma tabela para cada coordenada de amostras de RSS de alguns APs. Na fase de determinação, o nodo informa uma amostra do RSS que é comparada com cada entrada do tabela, o registro que mais se aproxima da amostra informada é adotada como atual posição do nodo.

Em [4] - *"Radar: an in-building RF-based user location and tracking system"*, Bahl e Padmanabhan propõem o método empírico, para localização baseada em RSS *indoor*. O método consiste em realizar uma série de medições da força de sinal das estações bases, e a partir dessas medições, a localização pode ser determinada fazendo uma triangulação dos dados coletados.

No artigo [4], para obter a posição de um nodo, são medidas as forças do sinal wifi de três *access points* (rss1, rss2, rss3), utiliza-se a distância Euclidiana, para fazer a comparação com os dados previamente coletados $sqrt((rss1 - rss1')^2 + (rss2 - rss2')^2 + (rss3 - rss3')^2)$, e assim encontrar o ponto que mais se aproxima dos parâmetros coletados naquele local.

A precisão da técnica de *fingerprinting* reside nas amostras de RSS, qualquer mudança no ambiente como reposição de APs, retirada ou inclusão de objetos e o trafego de pessoas no local podem interferir no desempenho desta técnica. Esse é o principal problema dessa técnica.

CAPÍTULO 4

IMPLEMENTAÇÃO

Este capitulo apresenta os detalhes de implementação do sistema proposto nesse trabalho, os testes realizados, os resultados obtidos e a análise dos mesmos. O aplicativo que implementa o sistema proposto, foi desenvolvido utilizando o Android SDK(*Software Development Kit*) que pode ser facilmente obtido em[11], ele foi totalmente escrito em Java e XML. O aplicativo é composto essencialmente por três classes: MainActivity, Map e WifiInterface.

A comunicação entre as três classes é feito através da classe Handler[12](disponibilizada no *framework*) que permite o envio de mensagens entre *threads* e componentes. As principais operações disponíveis no aplicativo não são executadas na *thread* principal, pois essas operações podem levar muito tempo para serem concluída, assim evitando o problema de travar a execução da *thread* principal, que já foi discutido anteriormente no capitulo sobre Android.

A figura 4.1 mostra como é feito a interação entre classes:

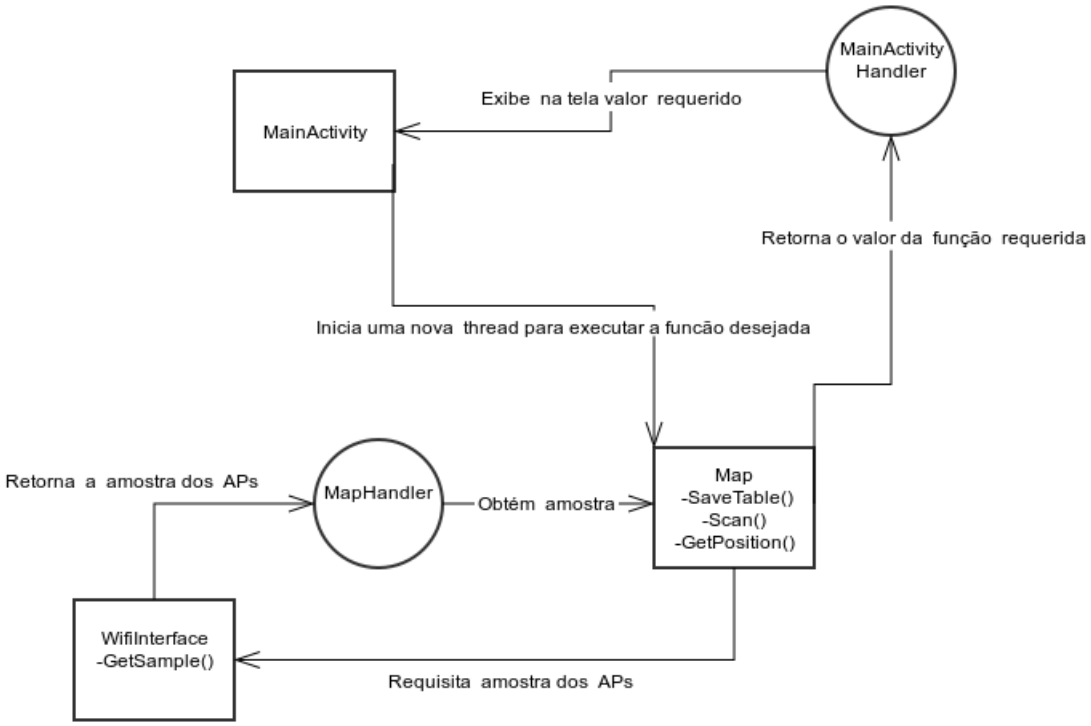

Figura 4.1: Esquema de interação entra as classes

4.1 MainActivity

A classe MainActivity é responsável por receber as entradas do usuário e exibir os resultados calculados. Ela também é responsável por instanciar as duas outras classes. Há três funções disponíveis:

- *Scan*: Obtém o *fingerprint* da força do sinal wifi dos APs na coordenada informada.
- *Save Table*: Salva a tabela de *fingerprints* em um arquivo.
- *Get Position*: Obtém a provável posição do aparelho.

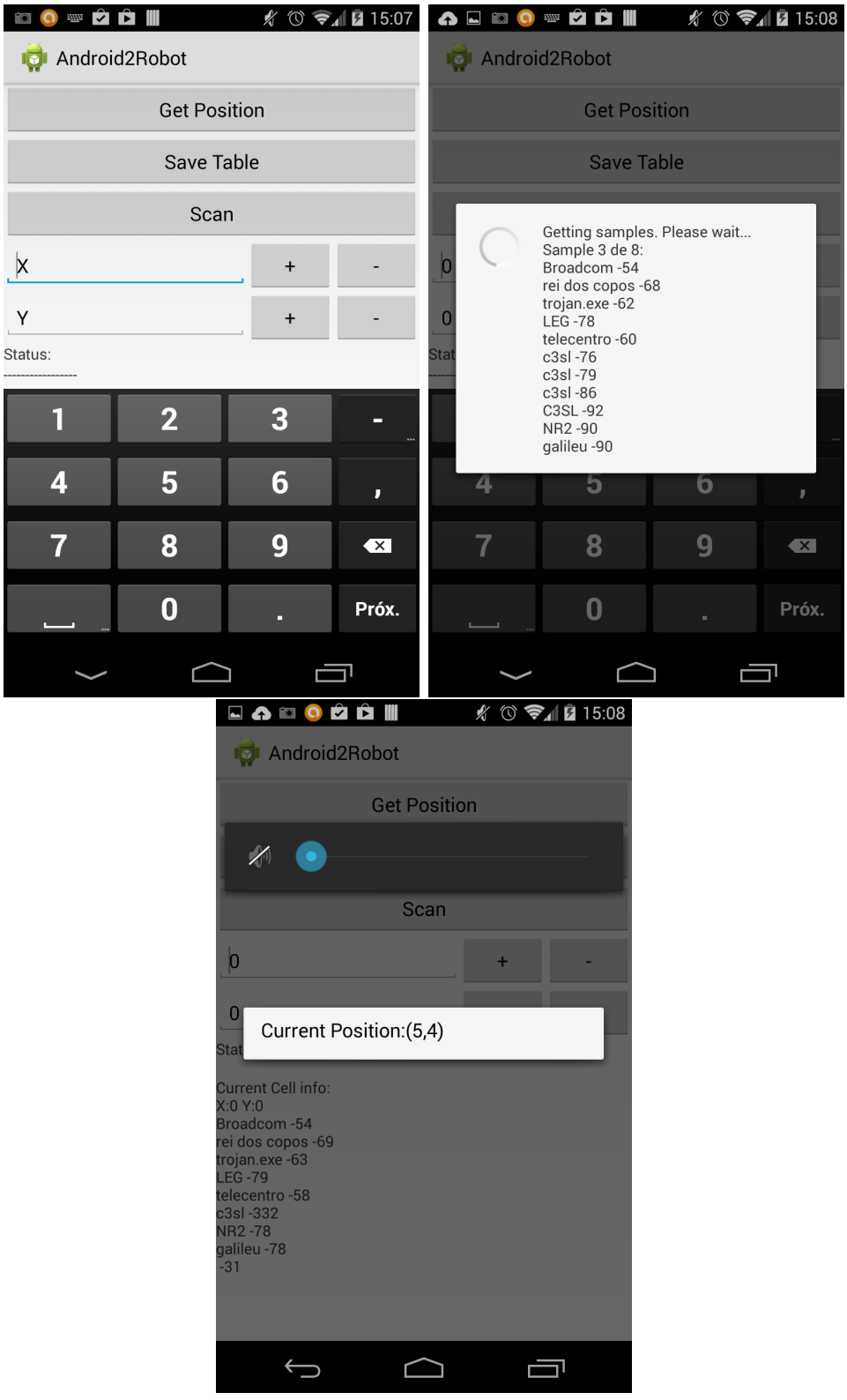

Figura 4.2: Tela inicial, tela na obtenção do fingerprint e tela da obtenção da posição corrente do aparelho.

A MainActivity cria novas *threads* para executar cada uma das operações requerida pelo o usuário.

4.2 WifiInterface

Esta classe centraliza todas as operações envolvendo wifi, como verificar e habilitar a conectividade com o wifi do aparelho e obter as informações dos APs(força do sinal, nome do AP, endereço MAC, etc) que estão no alcance do aparelho. Para obter essas informações, o objeto dessa classe instancia um BroadcastReceiver que escuta o modulo wifi do aparelho e repassa as informações obtidas através do *Handler* para o componente que requisitou elas.

4.3 Map

Esta classe é o núcleo da aplicação, ela carrega memória e armazena o mapa de *fingerprints* de RSS que estão guardadas em arquivo. O *fingerprints* é composto pela coordenada informada pelo usuário e as informações obtidas pela classe WifiInterface. Com o mapa de *fingerprints* essa classe executa o algoritmo descrito a seguir para disponibilizar a posição corrente do aparelho.

4.4 Algoritmo de Localização baseado em RSS *fingerprinting*

O algoritmo de localização aqui implementado é dividido em duas etapas: aprendizagem e determinação. Na etapa de aprendizagem, o aplicativo guarda o *fingerprint* dos APs da coordenada informada, e salva em uma tabela. O *fingerprint* é composto da média de 8 amostras do RSS de cada AP, a média é utilizada devido a oscilação da força do sinal medido pelo aparelho. Vale ressaltar que a quantidade de *fingerprints* coletados tem um grande impacto na precisão da localização como pode ser observado em [4].

Na segunda etapa, o processo de localização do aparelho é dividido em três fases:

1. Obtêm-se o *fingerprint* da localidade do aparelho com Android, da mesma forma feita na etapa de aprendizagem.

2. Calcula-se a distância euclidiana, para fazer a comparação com os dados previamente coletados $sqrt((rss1 - rss1')^2 + (rss2 - rss2')^2 + (rss3 - rss3')^2)$. A entrada com a menor distância euclidiana é escolhida como uma provável estimativa da posição do aparelho, esse processo é repetido para cada conjunto de três APs do *fingerprint* coletado na fase 1. Assim obtêm-se um conjunto de n pontos que são utilizados na próxima fase.

3. A estimativa da posição corrente do aparelho é obtida através do calculo do centroide $C(x_c, y_c)$ desses n pontos:

$$C\left(x_c, y_c\right) = \left(\frac{1}{n}\sum_{i=1}^{n} x_i \ , \ \frac{1}{n}\sum_{i=1}^{n} y_i\right)$$

Figura 4.3: Calculo de um centroide de um conjunto de pontos.

$C(x_c, y_c)$ minimiza a soma dos quadrados da distância euclidiana entre ele e os outros n pontos do conjunto[18].

Este algoritmo utiliza o preceito da correlação espacial das localidades adjacentes[5], ou seja, em uma determinada região do mapa, o RSS dos APs mesmo com alterações do ambiente, se mantém com valores similares. Tirando proveito disso, para estimar a posição do aparelho, obtém-se um conjunto de pontos que definem uma região de onde o aparelho possa estar, e a posição final do aparelho e obtida através do centroide desse conjunto de pontos, assim adicionando uma maior robustez a técnica de RSS *fingerprinting*.

4.5 Análise e Resultados Obtidos

Os testes foram realizados no primeiro andar do departamento de informática(DINF) da Universidade Federal do Paraná, devido a grande quantidade de APs disponíveis(até treze APs em uma coordenada), o que contribui para a precisão da técnica. O aparelho utilizado nos testes foi o *smartphone* LG Nexus 4 com Android KitKat versão 4.4.4.

4.5.1 Testes realizados

Na etapa de aprendizagem, para compor o mapa de *fingerprints*, foram coletados *fingerprints* de 67 locais distintos, onde o incremento de um no eixo x ou y significa um metro do primeiro andar do DINF. A imagem abaixo mostra os locais das amostras retiradas, marcados com quadrados pretos:

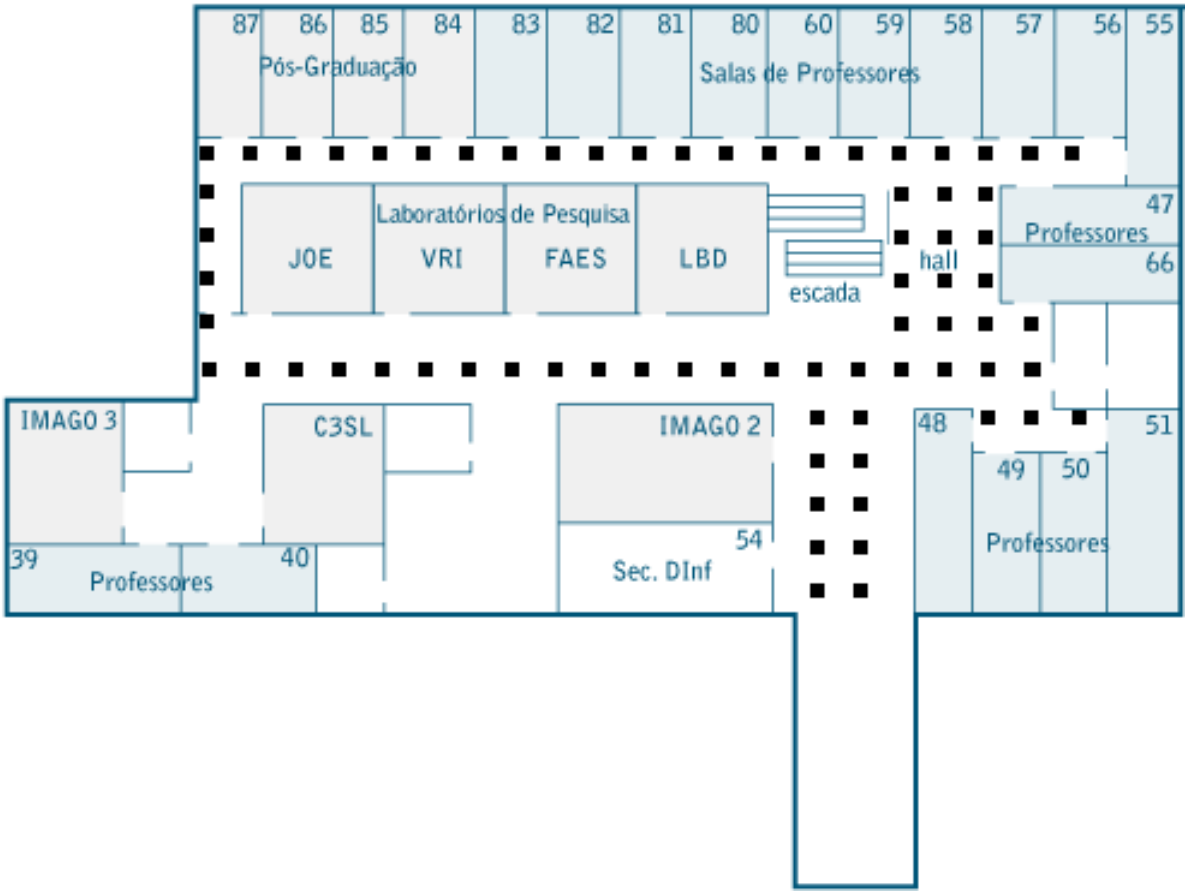

Figura 4.4: Mapa do primeiro andar do Departamento de Informática da UFPR.

Para testar a eficácia da aplicação na etapa de determinação, foram realizados 35 testes. A tabela abaixo mostra os resultados obtidos:

Teste	Estimativa(x,y)	Real(x,y)	Erro(metros)
1	(4, 0)	(4, 2)	2
2	(4, 0)	(4, 5)	5
3	(6, 0)	(7, 3)	3.16
4	(9, 3)	(8, 5)	2.24
5	(9, 8)	(9, 9)	1
6	(9, 9)	(9, 11)	2
7	(9, 10)	(12, 8)	3.61
8	(9, 12)	(9, 12)	0
9	(9, 13)	(7, 11)	2.83
10	(10, 14)	(9, 16)	2.24
11	(11, 15)	(13, 16)	2.24
12	(11, 15)	(10, 15)	1
13	(12, 15)	(12, 15)	0
14	(14, 15)	(12, 13)	2.83
15	(9, 16)	(9 ,14)	2
16	(9, 18)	(11, 20)	2.83
17	(8, 20)	(10, 19)	2.24
18	(10, 19)	(9, 19)	1
19	(7, 18)	(8, 18)	1
20	(6, 17)	(7, 18)	1.41
21	(6, 19)	(7, 18)	1.41
22	(6, 19)	(5, 17)	1
23	(4, 19)	(5, 16)	2.24
24	(4, 21)	(4, 21)	0
25	(4, 15)	(5, 16)	1.41
26	(4, 11)	(9, 12)	5.1
27	(4, 9)	(5, 12)	3.16
28	(4, 6)	(5, 8)	2.24
29	(4, 3)	(7 ,8)	5.83
30	(4, 2)	(5, 3)	1.41
31	(5, 0)	(7, 1)	2.24
32	(9, 1)	(6, 5)	5
33	(9, 7)	(7, 9)	2.83
34	(9, 7)	(7, 9)	2.83
35	(9, 5)	(7, 8)	3.6

Tabela 4.1: Tabela com os resultados obtidos.

Os resultados mostram que:

- 22,8% das estimativas possuem precisão de 1 metro ou menos.
- 77,1% das estimativas possuem precisão de 3 metros ou menos.
- 91,4 % das estimativas possuem precisão de 5 metros ou menos.

4.5.2 Análise dos Resultados Obtidos

Analisando os resultados obtidos nos testes realizados, pode-se concluir que o aplicativo implementado nesse trabalho, possui uma precisão satisfatória considerando um raio de 2 até 5 metros da posição real do aparelho.

Aparentemente, a oscilação do RSS e a correlação espacial das localidades adjacentes podem explicar a baixa precisão aguda obtida 22,8%(até 1 metro), pois com esses resultados, pode-se concluir que em um raio de até 5 metros em relação a posição real do aparelho, os *fingerprints* dessa região são muito semelhantes, o que dificulta uma precisão mais aguda da técnica.

Essa técnica pode ser bastante proveitosa na localização de robôs móveis, pois há a possibilidade de utilizar outros sensores neles disponíveis, para aumentar a precisão da técnica, incrementando o *fingerprint* com os dados coletados por esses sensores, e assim adicionando mais características para distinguir os *fingerprints*. Claro que deve ser considerado que o ambiente escolhido é bastante favorável a abordagem escolhida, dado o grande número de APs presentes.

CAPÍTULO 5

CONCLUSÃO

Este capitulo contém as conclusões e as possibilidades de trabalhos futuros referentes ao presente trabalho.

As técnicas de localização baseadas em RSS são muito promissoras e vem sendo bastante exploradas, isso graças a fácil aplicação e a grande quantidade de aparelhos disponíveis, com *hardware* capaz de medir RSS como *smartphones*, *tablets*, *notebooks* e *netbooks*.

Mas devido a natureza do RSS, há a necessidade da utilização de técnicas e algoritmos cada vez mais sofisticados, capazes de contornar as variações indesejados da força do sinal recebido, para se obter uma precisão satisfatória no processo de localização. O RSS oscila muito em função da interferência de obstáculos como paredes, móveis e outros objetos, trafego de pessoas, além de outros fatores como temperatura ambiente, umidade do ar, outros sinais, etc.

A localização baseada em RSS, é uma alternativa interessante para auxiliar robôs móveis no processo de estimar sua real posição em um ambiente *indoor*. A tendencia é que seja cada vez mais comum, a incorporação de robôs móveis no dia-a-dia dos seres humanos, auxiliando e desempenhando tarefas, também cada vez mais complexas, para que possam fornecer mais conforto, segurança e comodidade as pessoas.

5.1 Trabalhos Futuros

As possibilidades de trabalhos futuros são enormes. Primeiro, há a possibilidade de adicionar um maior precisão a técnica apresentada, por exemplo, levando em consideração as obstruções que interferem na força do sinal recebido, como o modelo de propagação de sinal proposto no artigo [4]. Nele são levados em consideração diversas variáveis para sua elaboração, sendo que a mais importante delas é a quantidade de obstáculos que estão entre o transmissor de sinal(*access point*) e o receptor (terminal móvel). Com base nesse

parâmetro, uma equação da distância em função da força de sinal recebido poderá ser deduzida, que será útil para os cálculos da posição e rastreamento do terminal móvel. Ou ainda combinar técnicas baseadas em RSS com as tecnologias já largamente empregadas, tais como o GPS.

Um outro problema importante do RSS *fingerprinting* que não é tratado pelo sistema proposto é o da atualização dinâmica da base de dados de *fingerprints*. Em [5] utiliza-se um modelo adaptativo para estimar o RSS de cada *access point* baseado na técnica de interpolação linear planar.

E por ultimo, fazer com que o aparelho com Android possa controlar um robô agindo como o cérebro do robô, onde este um se move e envia dados(via *bluetooth*) de seu sensores para o aparelho conforme for requerido e assim tomando as decisões necessárias para que o robô haja de maneira autônoma. Com essa combinação, pode-se fazer como no SLAM, onde os dados de RSS dos *access points* vão sendo coletados conforme o robô se locomove. A precisão da técnica proposta nesse trabalho pode ser aumentada, utilizando outros sensores do robô, pois um *fingerprint* pode ser composto por outros dados além do RSS dos *access points*, como informações coletadas por laser, sonar, câmera, etc.

BIBLIOGRAFIA

[1] Android developers. http://developer.android.com/index.html.

[2] What is android? http://wing-linux.sourceforge.net/guide/basics/what-is-android.html.

[3] Android. Android home page. http://www.android.com/.

[4] Paramvir Bahl e Venkata N.Padmanabhan. Radar: An in-building rf-based user location and tracking system. http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=832252&queryText%3DRADAR%3A+An+In-Building+RF-based+User+Location+and+Tracking+System, 2000.

[5] Kamol Kaemarungsi Chavalit Koweerawong, Komwut Wipusitwarakun. Indoor localization improvement via adaptive rss fingerprinting database. A. B. Smith-Jones, editor, *Information Networking (ICOIN), 2013 International Conference on*, páginas 412–416. IEEE, 2013.

[6] Paulo Roberto Godoi de Oliveira. Auto-localização e construção de mapas do ambiente para robôs móveis baseados em visão omniderecional estéreo. The Digital Library of Theses and Dissertations of the University of São Paulo, 2008.

[7] Android Developers. Android app component: Activities. http://developer.android.com/guide/components/activities.html.

[8] Android Developers. Android app component: Broadcast receiver. http://developer.android.com/reference/android/content/BroadcastReceiver.html.

[9] Android Developers. Android app component: Broadcast receiver. http://developer.android.com/guide/topics/connectivity/bluetooth.html.

[10] Android Developers. Android app component: Service. http://developer.android.com/guide/components/services.html.

[11] Android Developers. Get the android sdk. http://developer.android.com/sdk/index.html?hl=sk.

[12] Android Developers. Handler. http://developer.android.com/reference/android/os/Handler.html, 2010.

[13] M. W. M. Gamini Dissanayake, Paul Newman, Steven Clark, Hugh F. Durrant-Whyte, e M. Csorba. A solution to the simultaneous localization and map building (slam) problem. IEEE TRANSACTIONS ON ROBOTICS AND AUTOMATION, VOL. 17, NO. 3, 2001.

[14] The Apache Software Foundation. Apache license, version 2.0. http://www.apache.org/licenses/LICENSE-2.0, 2004.

[15] Khronos Group. The standard for embedded accelerated 3d graphics. http://www.khronos.org/opengles/.

[16] B. Hofmann-Wellenhof, H. Lichtenegger, e J. Collins. Global positioning system: Theory and practice. *Advances in Computer Science*. Springer Verlag, 2001.

[17] Landu Jiang. A wlan fingerprinting based indoor localization technique. http://digitalcommons.unl.edu/cgi/viewcontent.cgi?article=1059&context=computerscidiss, 2012.

[18] Roger A. Johnson. Advanced euclidean geometry. Dover, 2007.

[19] Chaomin Luo e Simon X. Yang. A bioinspired neural network for real-time concurrent map building and complete coverage robot navigation in unknown environments. IEEE TRANSACTIONS ON NEURAL NETWORKS, VOL. 19, NO. 7, 2008.

[20] Eduardo Marcel Macan. Níveis de execução. http://www.ccuec.unicamp.br/revista/infotec/linux/linux17-1.html, 2000. RIT - UNICAMP.

[21] Yongguo Mei, Yung-Hsiang Lu, Y. Charlie Hu, e C. S. George Lee. Deployment of mobile robots with energy and timing constraints. *IEEE TRANSACTIONS ON ROBOTICS, VOL. 22, NO. 3*. 2006.

[22] Luis Mengual, Oscar Marbán, antiago Eibe, e Ernestina Menasalvas. Multi-agent location system in wireless networks. `http://www.sciencedirect.com/science/article/pii/S095741741201161X`, 2013.

[23] Sung Wook Moon, Young Jin Kim, Ho Jun Myeong, Chang Soo Kim, Nam Ju Cha, e Dong Hwan Kim. Implementation of smartphone environment remote control and monitoring system for android operating system-based robot platform. *Ubiquitous Robots and Ambient Intelligence (URAI), 2011 8th International Conference on*. IEEE, 2011.

[24] Dragos Niculescu e Badri Nath. Ad hoc positioning system (aps) using aoa. IEEE INFOCOM 2003, 2003.

[25] N. Patwari, A. O. Hero III, e J. A. Costa. Learning sensor location from signal strength and connectivity.secure localization and time synchronization for wireless sensor and ad hoc network. Advances in Information Security series 30, 2006.

[26] Ana Carolina Fernandes Pena e Cláudio Elivan Santos da Silva. Serviços de localizaçã baseados em comunção móvel. `https://fenix.tecnico.ulisboa.pt/downloadFile/2589867775095/ENTREGA-Tese-Afonso_Vaz-54427_Final.pdf`.

[27] Tutorials Point. Android programing tutorial. `http://www.tutorialspoint.com/android/`.

[28] D. Scaramuzz R. Siegwart. Probabilistic map based localization. `http://www.asl.ethz.ch/education/master/mobile_robotics/year2011/Lecture_9.pdf`, 2011.

[29] D. Scaramuzz R. Siegwart. Probabilistic map based localization: Kalman filter localization. `http://www.asl.ethz.ch/education/master/mobile_robotics/year2011/Lecture_10.pdf`, 2011.

[30] D. Scaramuzz R. Siegwart. Slam:simultaneous localization and mapping. http://www.asl.ethz.ch/education/master/mobile_robotics/year2012/Lecture10.pdf, 2012.

[31] André Macêdo Santana. Localização e mapeamento simultâneos de ambientes planos usando visão monocular e representação híbrida do ambiente. http://bdtd.ufrn.br/tde_arquivos/19/TDE-2011-04-26T111523Z-3365/Publico/AndreMS_TESE_1-100.pdf, 2011.

[32] A. Savvides, C. Han, e M.B. Strivastava. Dynamic fine-grained localization in ad-hoc networks of sensors. MobiCom, 2001.

[33] Antonio Henrique Pinto Selvatici. Construção de mapas de objetos para navegação de robôs. *Advances in Computer Science*. The Digital Library of Theses and Dissertations of the University of São Paulo, 2009.

[34] stackoverflow. Get signal strength of wifi and mobile data. http://stackoverflow.com/questions/18399364/get-signal-strength-of-wifi-and-mobile-data, 2013.

[35] J Tourrilhes. Wireless tools for linux. http://www.hpl.hp.com/personal/Jean_Tourrilhes/Linux/Tools.html, 2010.

[36] Ze Wang, Yunlong Wang, Maode Ma, e Jigang Wu. Efficient localization for mobile sensor networks based on constraint rules optimized monte carlo method. dx.doi.org/10.1016/j.comnet.2013.06.010, 2013.

[37] Younggoo Kwon Youngbae Kong e Gwitae Park. Robust localization over obstructed interferences for inbuilding wireless applications. Consumer Electronics, IEEE Transactions, 2009.

[38] Xiaojun Zhu, Xiaobing Wu, e Guihai Chen. Relative localization for wireless sensor networks with linear topology. http://www.sciencedirect.com/science/article/pii/S014036641300176X, 2013.

CAPÍTULO 6

APÊNDICE A

Neste capitulo é apresentado o sistema operacional Android, pois o mesmo é proposto como base para os dispositivos que executarão a localização baseada em RSS *fingerprinting*.

O Android é um sistema operacional baseado no núcleo do Linux para dispositivos móveis, desenvolvido pela *Open Handset Alliance*, liderada pela Google e composta por outras empresas, tais como: Intel, Nvidia, Samsung, LG, entre outras [2]. O código fonte do sistema operacional está sobre a licença do Apache [14].

Ele apresenta vários recursos que podem facilmente empregados na robótica como câmeras, GPS, aceleradores gráficos 2D e 3D, *bluetooth*, wifi e também suporta diversos sensores como barômetro, magnetômetro, acelerômetro, sensor de proximidade, sensor de pressão, termômetro e compasso.

Este sistema operacional está presente em centenas de milhões de aparelhos em mais de 190 países [1] e possui um rico ambiente de desenvolvimento provido pelo Android SDK [11], incluindo um emulador de dispositivo, ferramentes de depuração, memória, performance e um *plugin* para o Eclipse (ADT). A seguir, são apresentadas a arquitetura desse sistema operacional e os componentes de uma aplicação Android.

6.1 Arquitetura

A arquitetura do Android é composta por cinco camadas, sendo elas: aplicações, *framework* de aplicações (*applications*), *android runtime*, bibliotecas (*libraries*) e kernel Linux. A figura abaixo mostra a disposição dessas camadas:

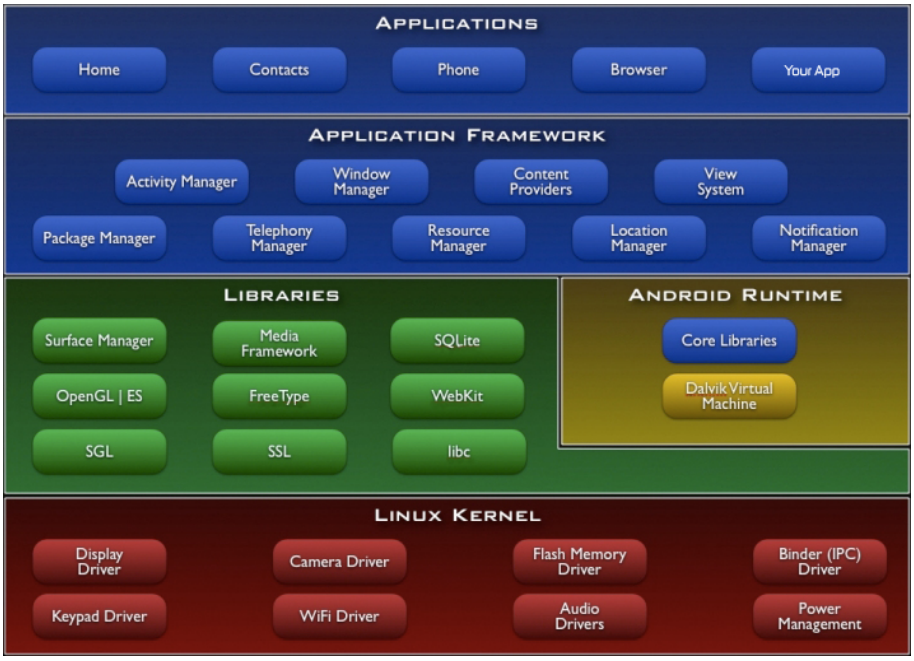

Figura 6.1: Arquitetura Android[27].

6.1.1 Aplicações

A camada de aplicações comporta todos aplicativos instalados no sistema operacional, a maioria delas escritas em java. Junto com o Android vai um conjunto de aplicações fundamentais. São elas: cliente de email, clente de SMS, agenda, gerenciador de contatos (Contacts), mapas e navegador (Browser).

6.1.2 *Framework* de Aplicações

Esta camada fornece vários componentes essenciais do sistema para as aplicações em forma de classes Java [27]. Eles gerenciam funções básicas do aparelho, os principais componentes disponíveis são:

- *Activity Manager*: gerencia o ciclo de vida de uma *activity* [7], explicado em mais detalhes na seção 4.2.1.
- *Content Providers*: faz o compartilhamento de dados entre aplicações.
- *Telephony Manager*: gerencia todas as ligações telefônicas.
- *Location Manager*: controla os sistemas de localização como GPS e torre celular.
- *Resource Manager*: fornece alguns recursos utilizadas por uma aplicação como *strings*, imagens, arquivos de *layout*, arquivos de áudio e vídeo.

A arquitetura de aplicações foi projetada para simplificar o reuso de componentes, qualquer aplicação publica suas funcionalidades e uma outra aplicação pode fazer uso delas (sujeito as regras de segurança impostas pelo *framework*) [2]. Essa estrutura permite que os componentes sejam substituídos pelo usuário.

6.1.3 Bibliotecas

O Android inclui um conjunto de bibliotecas nativas utilizadas por vários componentes do sistema. Elas foram implementada em C/C++, mas são acessadas através de interfaces Java [2] disponibilizada pelo *Framework* de aplicações. Abaixo, algumas das principais bibliotecas:

- *System C library*: uma implementação derivada da biblioteca C padrão sistema (libc) do BSD otimizada para dispositivos móveis.
- *Media Libraries*: baseado no PacketVideo's OpenCORE; as bibliotecas suportam os mais populares formatos de audio e video, bem como imagens estáticas.
- *Surface Manager*: gerencia o acesso a tela do aparelho bem como as múltiplas camadas de aplicações 2D e 3D.
- *LibWebCore*: *web browser engine* utilizado no Chrome(navegador padrão do Android).

- *3D libraries*: uma implementação baseada no OpenGL ES 1.0 [15]; as bibliotecas utilizam aceleração 3D via *hardware*; ou o *software* de renderização 3D altamente otimizado incluído no Android.
- SGL – o motor gráfico 2D.
- *FreeType*: faz renderização de fontes bitmap e vector.
- SQLite – um poderoso e leve *engine* de banco de dados.

6.1.4 *Android Runtime*

O Android inclui um grupo de bibliotecas que fornece a maioria das funcionalidades disponíveis nas principais bibliotecas da linguagem Java. Esta camada possui um componente chave do Android chamada máquina virtual Dalvik, ela é uma maquina virtual java baseada em registros especificamente projetada pra o Android [27] e otimizada para aparelhos que utilizam bateria e possuem memória e CPU limitados. Toda aplicação Android roda em seu próprio processo, com sua própria instância da máquina virtual Dalvik.

Apesar de que a maioria das aplicações Android serem escritas em Java, Java *byte code* não é suportado pela Dalvik. As classes Java são compiladas em arquivos .dex("Dalvik *executable*"), que são otimizados para consumo mínimo de memória, através da ferramenta "dx" incluída no SDK [2].

A Dalvik foi desenvolvido de forma a executar várias maquinas virtuais simultâneas eficientemente, fornecendo segurança e isolamento. O gerenciamento de memória e o suporte a *multi-threading* são feitos pelo kernel Linux.

A segurança entre aplicações e o sistema, é forçada pelos níveis de execução de processos do Linux[20] e cada aplicações ganha um id de usuário e grupo, um aplicativo só pode acessar um arquivo no qual pertença a seu usuário. Os aplicativos só podem acessar recursos do sistema nos quais forem previamente permitidos pelo usuário do aparelho na instalação do aplicativo.

Apesar de utilizar o Linux como base, a portabilidade de aplicativos Linux para Android não é simples, pois ele não possui o sistema de janelas X, nem suporte completo ao

conjunto padrão de bibliotecas GNU.

6.1.5 Linux Kernel

A camada base da arquitetura do Android é o kernel Linux com algumas alterações feitas pela Google. As versões correntes do sistema operacional utilizam o kernel Linux 3.x como base, enquanto que os dispositivos com a versão do sistema operacional inferior a o 4.0 (*Ice Cream Sandwich*) são baseados no kernel Linux 2.6.x [27].

O kernel Linux é responsável por executar os serviços centrais do sistema, onde o Linux é realmente bom, como segurança, gerenciamento de processos, memória e rede. Ele também fornece suporte a uma vasta quantidade *drivers* essenciais para *hardwares* periféricos como câmera, teclado, tela, *bluetooth*, facilitando a portabilidade do Android para diferentes tipos de *hardwares*. Assim, essa camada faz a interface com o hardware do aparelho.

6.2 Componentes de uma Aplicação

O código java compilado junto com outros recursos requeridos pela aplicação, são empacotados em um único arquivo com extensão .apk (*"Android Package"*), ele é quem será instalado no aparelho. Tudo que esta contido neste arquivo é considerado uma aplicação Android [2].

Um característica interessante do Android é que uma aplicação pode utilizar componentes de outra aplicação (desde que esta aplicação permita), sem que seja necessário incorporar o código ou criar um *link* para este componente, basta inicializa-lo quando necessário.

Para que isso funcione, o sistema deve ser capaz de inciar um processo quando esse componente é requirido, e também instanciar todos os objetos java desse componente. Ao contrario de aplicações de outros sistemas, as aplicações Android não possui um único ponto de inicialização (não há função main(), por exemplo). Elas possuem componentes essenciais que podem ser inicializados e executados sempre que forem necessários. Há

quatro tipos desses componentes: *activity*, *service*, *broadcast receiver* e *content provider*, explicados a seguir.

Quando um componente precisa ser executado, o Android inicia um processo Linux com uma única *thread* (*thread* principal). Por padrão, todos os componentes de uma aplicação executam nesse processo e nessa *thread*, mas se necessário, o sistema permite que componentes executem em outros processos, e novas *threads* possam ser instanciadas por qualquer processo.

As chamadas ao sistema são feitas pela *thread* principal. Como todos os componentes executam nessa *thread*, eles não devem realizar operações que levem muito tempo para serem finalizadas (como a computação de laços ou operações envolvendo internet), pois isso pode bloquear a execução de outros componentes do processo. Operações que demandam de muito tempo devem ser executadas em uma nova *thread*.

Cada componente tem suas características e comportamentos distintos, a seguir serão vistos em mais detalhes cada tipo de componente.

6.2.1 Activity

Uma *activity* é um componente da aplicação que fornece uma tela na qual o usuário possa interagir [7]. Para cada *activity* é dada uma janela na qual ela possa exibir seu elementos, uma janela pode ou não preencher toda a tela.

Uma aplicação frequentemente possui varias *activities* que são ligadas umas ás outras. Tipicamente, uma *activity* é especificada como principal, ou seja, ela que exibida quando o usuário executa a aplicação pela primeira vez. Cada *activity* pode iniciar outra *activities* pra realizar uma outra tarefa. Toda vez que uma nova *activity* é iniciada e apresentada(*foreground*) ao usuário, a *activity* anterior é "parada" e o sistema insere ela em uma pilha. Quando o usuário pressiona o botão voltar, uma *activity* é desempilhada e devolvida para a tela do usuário, e *activity* anterior é destruída.

O componente *activity* possui um ciclo de vida especial, definido por um conjunto de estados implementado por um sistema em cascata de chamada de métodos específicos. Abaixo o esquema de transição desses estados:

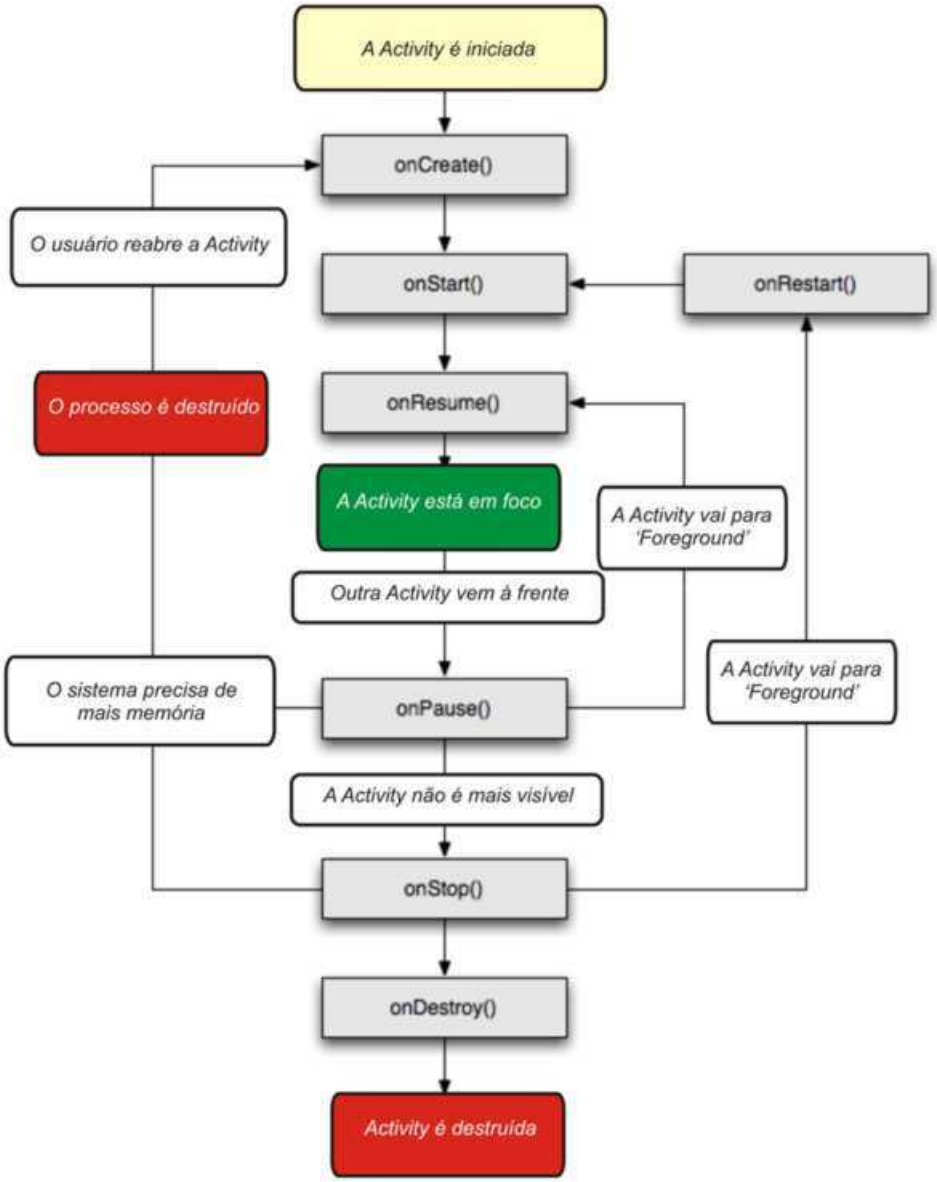

Figura 6.2: Ciclo de vida de uma *activity*[27].

Os métodos que representam os estado do ciclo de vida tem as seguintes funções:

- **onCreate():** é chamado uma única vez, este método é responsável por construir a tela que será apresentada.
- **onStart():** é chamado quando a *activity* está prestes a ficar visível para o usuário.
- **onResume():** é chamado quando a *activity* é visível para o usuário.
- **onRestart():** este método é chamado quando a *activity* foi temporariamente parada e está sendo reiniciada.

- **onPause():** é chamado quando a *activity* esta sendo encerrada, e não está mais visível ao usuário.
- **onStop():** é chamado quando a *activity* está sendo encerrada.
- **onDestroy():** este encerra a execução de uma *activity*.

6.2.2 Services

Um *service* não possui interface visual, pois sua função é executar tarefas em segundo plano por um período de tempo indefinido [10], mesmo que o usuário troque de aplicação o *service* continua sua execução. Por exemplo, um *service* pode tocar uma musica em segundo plano, buscar alguma informação na rede ou calcular algo e entregar o resultado para uma *activity* que precise.

Os *services* disponibilizam uma interface que permite que mesmo em execução outros componentes possam se comunicar com ele, isso possibilita fazer comunicações entre processos [10]. Por exemplo, para um *service* responsável por tocar uma lista de musicas, isso seria útil, pois permitiria que o usuário pausasse, parasse e recomeçasse a execução da lista.

Como os outros componentes *services* executam na *thread* principal. Então para não bloquear outros componentes ou a interface do usuário, frequentemente eles inciam outra *thread* para realizar tarefas que levam muito tempo para serem concluídas.

6.2.3 Broadcast Receiver

Broadcast receiver é um componente que recebe e responde a notificações [8]. Muitas dessas notificações são emitidas pelo sistema operacional como mudança de fuso horário, carga da bateria está baixa, uma foto foi retirada ou o usuário mudou a linguagem corrente. Qualquer aplicações pode também emitir notificações.

Uma aplicações pode ter diversos *broadcast receivers* e responder a várias notificações que ela considere importante. Todos os *receivers* herdam da classe BroadcastReceiver. Tipicamente eles não fazem nenhum processamento sobre a informação recebida, um

broadcast receivers deve iniciar uma *activity* ou um *service* para responder a notificação recebida.

6.2.4 Content Providers

Um *content provider* faz com que um conjunto de dados da aplicação fique disponível para outra aplicações [9]. Eles são a única maneira de compartilhar dados entre aplicações. Ou seja, para tornar os dados públicos, só há duas maneiras: criando um *content provider* (uma subclasse ContentProvider) ou adicionando os dados em provedor existente - se existe um que controle o mesmo tipo de dados e se tenha permissão para escrever nele.

O *content provider* criado a partir da classe ContentProvider, implementa um conjunto de métodos que possibilita outras aplicações acessar e salvar dados do tipo que ele controla. No entanto, as aplicações não acessam esses métodos diretamente. Para isso, elas devem utilizar um objeto da classe ContentResolver. Onde este pode se comunicar com qualquer *content provider*, eles cooperam para gerenciar a comunicação entre processos envolvidos.

O Android fornece um conjunto de *content providers* para dados comuns como áudio, vídeo, imagens e informações dos contatos. As aplicações podem acessar esses *providers* para obter os dados que eles contem, para alguns é necessário obter permissões especias para ler os dados.

Os dados podem ser armazenados no sistema de arquivos ou em um banco de dados SQLite. *Content providers* apresentam seus dados como uma tabela de um banco de dados, onde cada linha é um registro e cada coluna é um dado de um tipo particular.

Em resumo este capitulo apresentou as principais características do sistema operacional base para o desenvolvimento do sistema de localização proposto no presente trabalho.