EDILEUTON HENRIQUE DE OLIVEIRA

LOCALIZAÇÃO *INDOOR* PARA ROBÔS MÓVEIS

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: Eduardo Todt.

CURITIBA

2013

SUMÁRIO

RESUMO **iii**

1 INTRODUÇÃO **1**

1.1 Objetivo 3

1.2 Organização do trabalho 3

2 ROBÔS MÓVEIS **4**

2.1 Localização 4

2.1.1 Localização de Markov 5

2.1.2 Localização filtro de Kalman 6

2.2 Construção de Mapas 6

2.2.1 Mapas Métricos 7

2.2.2 Mapas Topológicos 7

2.3 Problema de auto localização e mapeamento simultâneos 8

3 *WIRELESS SENSOR NETWORKS* - WSNS **10**

3.1 TOA(*Time of Arrival*) 10

3.2 TDOA(*Time Difference of Arrival of Two Different Signals*) 11

3.3 AOA(*Angle of Arrival*) 12

3.4 RSS(*Received Signal Strength*) 13

4 ANDROID **15**

4.1 Arquitetura 16

4.1.1 Aplicações 16

4.1.2 *Framework* de Aplicações 16

4.1.3 Bibliotecas 17

4.1.4 *Android Runtime* 18

4.1.5 Linux Kernel 19

4.2 Componentes de uma Aplicação 19
4.2.1 Activity 20
4.2.2 Services 20
4.2.3 Broadcast Receiver 21
4.2.4 Content Providers 21

5 IMPLEMENTAÇÃO **23**

6 CONCLUSÃO **24**

6.1 Trabalhos Futuros 24

BIBLIOGRAFIA **27**

RESUMO

CAPÍTULO 1

INTRODUÇÃO

Robôs móveis são sistemas incorporados no mundo real que se movem autonomamente e interagem com ele para realizar suas tarefas[26]. 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 [14].

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 [26]. Tipicamente, existem duas abordagens para a representação de mapas de ambiente [18]: 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 [18].

Na navegação com mapas topológicos, utiliza-se 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 [18].

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 [13], 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 calculo da localização depende das informações de localização dos nodos(que podem ser estáticos ou móveis) da rede[27]. 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) [12], ângulo de chegada (*Angle of Arrival* - AOA) [17], diferença de chegada de dois sinais diferentes (*Time Difference of Arrival of Two Different Signals* - TDOA) [25] e força do sinal recebido (*Received Signal Strength* - RSS) [10], são alguns das abordagens feitas por algoritmos que utilizam WSNs, para calculo da localização.

Este trabalho trás uma proposta de localização *indoor* baseado em RSS para robôs móveis. Utilizando como base o método empírico sugerido no artigo[10]. Nesse trabalho é utilizada a plataforma Android, pois ele [9] é uma plataforma de fácil desenvolvimento, com documentação farta [8], e possui *drivers* para câmera, wifi, acelerômetro, *bluetooth*, microfone, compasso e GPS [16].

1.1 Objetivo

Implementar um sistema de localização *indoor* baseado em RSS para robôs móveis, utilizando a plataforma Android. Tendo como base o método empírico proposto por Bahl e Padmanabhan no artigo [10], aplicando o método dos quadrados mínimos para encontrar o posição que mais se aproxima da posição real do robô.

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

ROBÔS MÓVEIS

Neste capitulo serão discutidas algumas das principais tarefas que o robô deve desempenhar, para que ele possa concluir seu objetivos 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 calculo 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 [21]. 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 nos da a distribuição de probabilidade de todas as possíveis configurações do robô [21]. 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 [21]:

- 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 calculo 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* contêm a probabilidade de robô estar nela [21]. A distribuição de probabilidade associada à percepção dos sensores também é discretizado. 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[21].

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 gaussiana [22]. 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.

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.

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 [26]. 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[26]. Há duas abordagens principais para a representação de mapas de ambiente [18]. 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 [26].

Com à alta densidade de informação nesses mapas, o resultado da localização é mais preciso e menos sujeito a ambiguidades [26]. 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 assumem 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, de forma geral, registram informações sobre determinados elementos ou locais do ambiente, chamados marcos [26]. Assim, marcos e relações são os elementos dos mapas topológicos. Essas relações podem ser de vários tipos, tais como o deslocamento

relativo entre dois marcos, a existência de um caminho entre eles, quantidade de energia gasto em um caminho entre marcos,etc. 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 [26].

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 [11].

O SLAM tem sido tema de várias pesquisas na área de robótica. A grande vantagem do SLAM é que elimina a necessidade um conhecimento topológico 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) [23]. 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[11].

O EKF SLAM resume toda a toda experiencia obtida pelo robô em um vetor de estados estendido Y, compreendendo a pose do robô, a posição dos marcos do mapa e a matriz de covariância P[23]. 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 são usados 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[18].

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) [23]. 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 [24].

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[18], ela é chamada de GraphSLAM. As posições do robô ao longo do tempo e os marcos correspondem a nós em um grafo [23]. 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

WIRELESS SENSOR NETWORKS - WSNS

Este capitulo aborda as principais técnicas de localização utilizando WSNs.

WSNs 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 [27]. Localização é um tópico muito importante, pois, muitas aplicações das WSNs dependem do conhecimento das posições dos sensores. 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 calculo da localização depende das informações de localização dos nodos(que 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* [27] [29] 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[12]. 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 [12], ou seja, 1 ms corresponde a 0,3 m na propagação sonora, enquanto para a rádio frequência,

1 ns corresponde a 0,3 m [12].

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 of Two Different Signals*)

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)) [25]. 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 [25].

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 por trás 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 recta e, a intersecção das duas linhas, vai fornecer a posição do dispositivo[20].

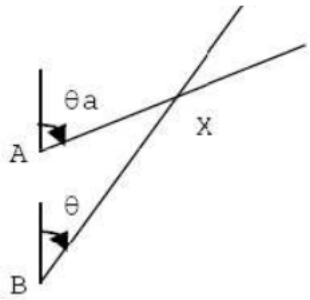

Figura 3.1: Anglel of Arrival[20].

Existem duas maneiras nas quais sensores medem o AOA [17]. 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 ser uma técnica de fácil aplicação.

O funcionamento da RSS é baseado na medição do sinal no receptor e o valor obtido indica a distância até o transmissor [19]. 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 [28]:

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

Figura 3.2: Modelo Log-normal [28].

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 [10][19][29][15] [28].

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

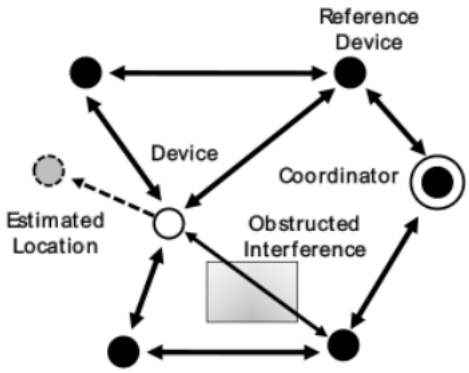

Figura 3.3: Influência de um obstáculos na localização de um nodo[19].

Em [10] - *"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 [10], para obter a posição de um nodo, são medidas as forças do sinal wifi de 3 *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.

CAPÍTULO 4

ANDROID

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 [1]. O código fonte do sistema operacional está sobre a licença do apache[2].

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.

O sistema operacional está presente em centenas de milhões de aparelhos em mais de 190 países [8] e possui um rico ambiente de desenvolvimento provido pelo Android SDK [3], incluindo um emulador de dispositivo, ferramentes de depuração, memória, performance e um *plugin* para o Eclipse (ADT). A seguir, serão apresentadas a arquitetura desse sistema operacional e os componentes de uma aplicação Android.

4.1 Arquitetura

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

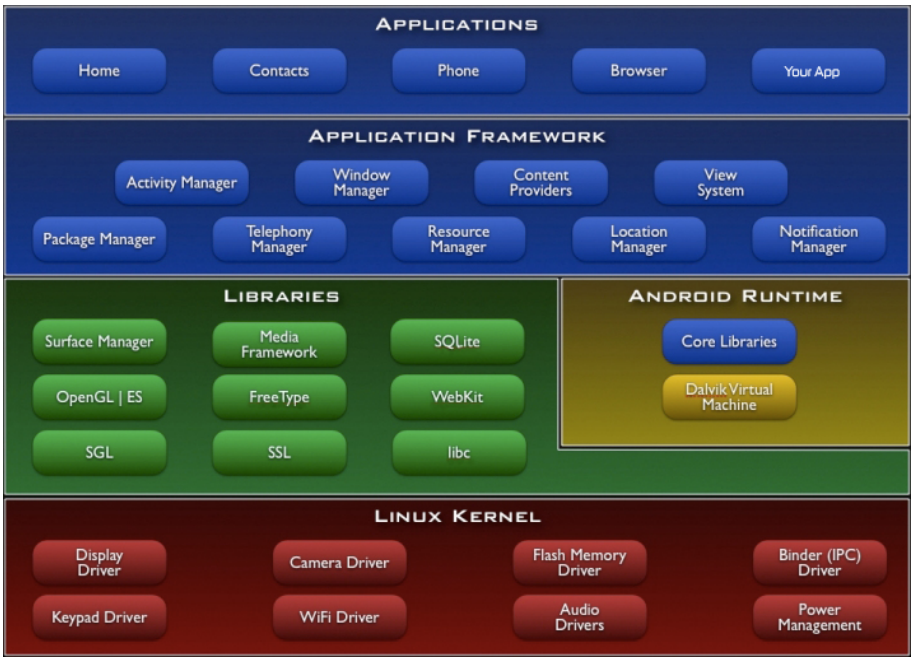

Figura 4.1: Arquitetura Android[4].

4.1.1 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, mapas e navegador.

4.1.2 *Framework* de Aplicações

Esta camada fornece vários componentes essenciais do sistema para as aplicações em forma de classes Java [4]. Eles gerenciam funções básicas do aparelho, os principais componentes disponíveis são:

- *Activity Manager*: gerencia o ciclo de vida de uma *activity* [7].

- *Content Providers*: faz o compartilhamento de dados entre aplicações.
- *Telephony Manager*: gerencia todas as ligações.
- *Location Manager*: controla os sistemas de localização como GPS e torre celular.
- *Resource Manager*: fornece alguns recursos utilizadas por uma aplicação como *strings*, imagens e arquivos de *layout*.

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*) [1]. Essa estrutura permite que os componentes sejam substituídos pelo usuário.

4.1.3 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 [1] 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 [5]; 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.

4.1.4 *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 [4] 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 [1].

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[6] 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.

4.1.5 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 [4].

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.

4.2 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.

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*.

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.

4.2.1 Activity

Uma *activity* é um componente da aplicação que fornece uma tela na qual o usuário possa interagir. 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 ligada umas as 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 apresenta 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 desempilha e devolvida para a tela do usuário, e *activity* anterior é destruída.

4.2.2 Services

Um *service* não possui interface visual, pois sua função é executar tarefas em segundo plano por um período de tempo indefinido, mesmo que o usuário troque de aplicação o *service* continua sua execução. Por exemplo, um *service* pode tocar uma musica em segundo, 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. 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.

4.2.3 Broadcast Receiver

Broadcast receiver é um componente que recebe e responde a notificações. 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.

4.2.4 Content Providers

Um *content provider* faz com que um conjunto de dados da aplicação fique disponível para outra aplicações. 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 exite 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 audio, video, 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.

CAPÍTULO 5

IMPLEMENTAÇÃO

CAPÍTULO 6

CONCLUSÃO

6.1 Trabalhos Futuros

BIBLIOGRAFIA

[1]

[2]

[3]

[4]

[5]

[6]

[7] Android app component: Activities.

[8] Android developers.

[9] Android. Android home page.

[10] Paramvir Bahl e Venkata N.Padmanabhan. Radar: An in-building rf-based user location and tracking system. IEEE INFOCOM 2000, 2000.

[11] 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.

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

[13] 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.

[14] 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.

[15] Luis Mengual, Oscar Marbán, antiago Eibe, e Ernestina Menasalvas. Multi-agent location system in wireless networks. Expert Systems with Applications, 2013.

[16] 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.

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

[18] PAULO ROBERTO GODOI DE OLIVEIRA. Auto-localizaÇÃo e construÇÃo de mapas de ambiente para robÔs mÓveis baseados em visÃo omnidirecional estÉreo. The Digital Library of Theses and Dissertations of the University of São Paulo, 2008.

[19] 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.

[20] Ana Carolina Fernandes Pena e Cláudio Elivan Santos da Silva. Serviços de localizaçã baseados em comunção móvel.

[21] D. Scaramuzz R. Siegwart. Probabilistic map based localization.

[22] D. Scaramuzz R. Siegwart. Probabilistic map based localization: Kalman filter localization.

[23] D. Scaramuzz R. Siegwart. Slam:simultaneous localization and mapping.

[24] André Macêdo Santana. Localização e mapeamento simultâneos de ambientes planos usando visão monocular e representação híbrida do ambiente.

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

[26] 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.

[27] Ze Wang, Yunlong Wang, Maode Ma, e Jigang Wu. Efficient localization for mobile sensor networks based on constraint rules optimized monte carlo method. Computer Networks, 2013.

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

[29] Xiaojun Zhu, Xiaobing Wu, e Guihai Chen. Relative localization for wireless sensor networks with linear topology. Computer Communications, 2013.