UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
CURSO DE GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Jéferson Farias da Silva

SISTEMA PARA AVISO DE APROXIMAÇÃO ENTRE VEÍCULO E PONTO ESTÁTICO

Santa Maria, RS
2016

Jéferson Farias da Silva

SISTEMA PARA AVISO DE APROXIMAÇÃO ENTRE VEÍCULO E PONTO ESTÁTICO

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação, da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de **Bacharel em Ciência da Computação**.

ORIENTADORA: Prof.ª Marcia Pasin

Trabalho de Graduação Nº 426
Santa Maria, RS
2016

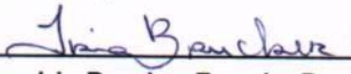
Jéferson Farias da Silva

SISTEMA PARA AVISO DE APROXIMAÇÃO ENTRE VEÍCULO E PONTO ESTÁTICO

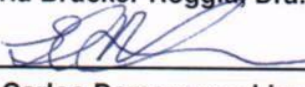
Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de
Bacharel em Ciência da Computação.

Aprovado em 15 de dezembro de 2016:

Marcia Pasin, Dra. (UFSM)
(Presidente/Orientadora)



Iria Brucker Roggia, Dra. (UFSM)



João Carlos Damasceno Lima, Dr. (UFSM)

Santa Maria, RS
2016

AGRADECIMENTOS

Primeiramente agradeço ao meu pai Pedro O. R. da Silva, e minha mãe Marly F. da Silva pelo apoio constante e por tudo que sempre fizeram por mim durante os anos de graduação, pois não mediram esforços para que eu chegasse a essa etapa de minha vida.

A minha Professora orientadora Drª. Marcia Pasin, pela disponibilidade de materiais e inestimável auxílio em todas as etapas de realização deste trabalho. Agradeço à Professora Drª. Iria Brucker Roggia e ao Professor Dr. João Carlos Damasceno Lima por terem aceitado o convite para participarem da banca examinadora, e a todos os professores do curso de Ciência da Computação, que contribuíram e enriqueceram meus conhecimentos durante minha vida acadêmica.

Agradeço à Universidade Federal de Santa Maria, por me proporcionar uma singular formação no ensino superior.

Aos amigos e familiares, pelo apoio, presença e incentivo constantes.

E a todos que de alguma forma contribuíram para a minha chegada nesta etapa.

Você acredita que é a sua mente. Eis aí o delírio. O instrumento se apossou de você.

(Eckhart Tolle)

RESUMO

SISTEMA PARA AVISO DE APROXIMAÇÃO ENTRE VEÍCULO E PONTO ESTÁTICO

AUTOR: Jéferson Farias da Silva
ORIENTADORA: Marcia Pasin

Este trabalho apresenta uma solução para inferir a posição de um veículo em relação a pontos estáticos usando a plataforma robótica Arduino. A plataforma Arduino é uma solução de *software* e *hardware* com especificação aberta e de baixo custo, e permite investigar e avaliar novos serviços antes de disponibilizá-los na prática. O cenário em questão é encontrado em muitas situações cotidianas no trânsito. Exemplos incluem aproximação de um ônibus a pontos de paradas, e aproximação de veículos a um conjunto de semáforos. A comunicação entre o veículo (robô Arduino) e os pontos estáticos (simulados por computadores convencionais) é realizada via rede *Wi-Fi* (*Wireless Fidelity*), sem a necessidade do uso de ferramentas de GPS (*Global Positioning System*). A posição dos pontos estáticos é conhecida e o veículo desloca-se seguindo um percurso pré-estabelecido. A posição do veículo em relação aos pontos estáticos é estimada unicamente pela intensidade do sinal da rede *Wi-Fi*.

Palavras-chave: Localização. Mapeamento. Trânsito. Arduino. Simulação. *Wi-Fi.*

ABSTRACT

APPROACH NOTICE SYSTEM BETWEEN VEHICLE AND STATIC POINT

AUTHOR: Jéferson Farias da Silva
ADVISOR: Marcia Pasin

This work presents a service to infer the positioning of a vehicle in relation to static points using the Arduino robotic platform. The Arduino platform is an open-source, low-cost software and hardware solution that allows to investigate and evaluate new services before making them available in practice. The scenario in question can be found in many everyday situations in traffic. Examples include approaching a bus at stop points, and approaching vehicles at a set of traffic lights. The communication between the vehicle (Arduino robot) and the static points (simulated by conventional computers) is carried out via Wi-Fi network without the need for GPS tools. The position of the static points is known and the vehicle moves following a pre-established route. The position of the vehicle relative to the static points is estimated solely by the use of signal strength of the Wi-Fi network.

Keywords: Location. Traffic. Mapping. Arduino. Simulation. Wi-Fi.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AC	*Alternating Current*
AHLos	*Ad-Hoc Localization System*
ATIS	*Advanced Traveler Information Systems*
ATMS	*Advanced Transportation Management Systems*
AVCS	*Advanced Vehicle Control Systems*
APTS	*Advanced Public Tansportation Systems*
CVO	*Commercial Vehicle Operations*
DC	*Direct Current*
DBm	Decibéis por Minuto
EEPROM	*Electrically Erasable Programmable Read Only Memory*
ETTM	*Eletronic Toll and Traffic Management*
GPS	*Global Positioning System*
ID	*Identity*
IDE	*Integrated Development Environment*
IP	*Internet Protocol*
ITS	*Inteligent Transportation System*
MAC	*Media Access Control*
P2P	*Peer-to-Peer*
RAM	*Random Access Memory*
RISC	*Reduced Instruction Set Computer*
ROM	*Read Only Memory*
SDK	*Software Development Kit*
SRAM	*Static Random Access Memory*
SSID	*Service Set Identifier*
TCP	*Transmission Control Protocol*
USB	*Universal Serial Bus*
VANET	*Vehicular Ad-Hoc Network*
WAVE	*Wireless Access in a Vehicular Environment*
WEB	*World Wide Web*
WEP	*Wired Equivalent Privacy*
Wi-Fi	*Wireless Fidelity*
WPA	*WiFi Protected Access*

SUMÁRIO

1 INTRODUÇÃO

O crescente aumento no número de veículos em trânsito causa diversos problemas, como congestionamentos, aumento no número de acidentes e atrasos em serviços de transporte. Diversas tecnologias estão sendo empregadas nos veículos para facilitar a dirigibilidade e garantir a interação entre veículo e condutor, como por exemplo, sensores para detecção de obstáculos no ambiente e sistemas de GPS que informam dados de forma constante sobre o trajeto. Ferramentas desse tipo facilitam o uso individual do veículo mas não influenciam de forma direta no meio coletivo. Para que isso ocorra é necessária a colabração entre os veículos e também entre veículos e infraestruturas através de canais de comunicação, formando assim as chamadas redes veiculares. Tais redes são compostas por sistemas de transporte inteligentes (*Intelligent Transportation System - ITS*), que visam monitorar e gerenciar fatores como o fluxo do trânsito, definir rotas para melhorar a segurança, reduzir o desgaste do veículo, reduzir os tempos de viagem e reduzir consumo de combustível (BIEM et al., 2010).

Uma rede veicular, ou VANET (*Veicular Ad Hoc Network*), que é uma tecnologia existente como uma subclasse das redes *Ad Hoc* (comunicação direta - ponto a ponto), tem como objetivo a comunicação entre veículos e/ou entre veículos e uma infraestrutura de acostamento tendo a finalidade de proporcionar segurança e autonomia nas vias rodoviárias (BARBIERI, 2012). Para a camada física de dispositivos móveis que implementam VANET's, existe o padrão WAVE (*Wireless Access in a Vehicular Environment*) que pode ser dividido em duas modalidades de aplicação: Aplicações de segurança e aplicações de não-segurança (HARTENSTEIN; LABERTEAUX, 2008). As aplicações de segurança são voltadas para auxílio em momentos críticos, como por exemplo em pontos de cruzamento e ultrapassagens. Já as aplicações de não segurança visam melhorar a experiência do passageiro ou pedestre, apresentando formas de entretenimento, aplicativos, informações sobre o trânsito e pontos de interesse (por exemplo, hotéis, restaurantes, entre outros).

Este trabalho apresenta uma base para uma aplicação de não segurança, trazendo uma forma de detectar a aproximação entre veículo e ponto estático, via internet *Wi-Fi*, e para avaliar o serviço, será implementada uma prototipação utlizando a plataforma Arduino, que é destinada a artistas, designers, hobbistas e qualquer pessoa interessada em criar objetos ou ambientes interativos (ARDUINO, 2009). Além das aplicações específicas de trânsito, vislumbra-se o acesso a Internet em qualquer lugar e a qualquer instante (LI; WANG, 2007), então há a possibilidade de criação de um canal alternativo para troca de informações nesse meio, que pode ser usado por aplicações que necessitam de sinais simples ou como meio de reserva ou assistência, em caso de falha em pontos da VANETs. O cenário aqui abordado é a troca de

sinais entre um ônibus (estrutura móvel) e um ponto de parada (estrutura estática), representados respectivamente por um carro Arduino e computadores convencionais, todos conectados a mesma rede *Wi-Fi*. Os sinais são usados para determinar sobre qual estrutura estática o veículo está mais próximo.

Tomando como foco os meios de transporte, sabe-se que as cidades brasileiras não foram estruturadas a partir da perspectiva de que a maioria dos deslocamentos de passageiros fossem feitos por automóveis. A participação histórica dos deslocamentos por automóvel ficava em torno de 25% em relação ao total dos deslocamentos urbanos (VACCARI; FANINI, 2010), o que não acontece atualmente. Para quem possui veículo próprio, torna-se mais fácil o deslocamento com esse meio devido aos desconfortos que o transporte público apresenta, como por exemplo, dificuldade de obter informações a respeito de linhas e horários. Avançar na qualidade e na eficiência dos transporte público urbano é importante para estimular a população e melhorar a mobilidade urbana.

Apresentar e implementar meios de informação é a base para a melhorar as redes do transporte público, porque evita transtornos e facilita a tomada de decisão do usuário, sobre, por exemplo, qual linha utilizar para chegar ao destino e em que horário. Há cidades que possuem o sistema de transporte integrado (como por exemplo, Santa Maria e Porto Alegre), que permite ao passageiro utilizar duas linhas de ônibus, em um certo período de tempo, pagando apenas uma passagem. Caso o usuário possua um meio de informação rápido e de fácil interpretação, pode avaliar a possibilidade de utilizar duas linhas de transporte alternativas, ao invés de aguardar por um longo período de tempo pela sua linha usual. Esse tipo de uso torna-se especialmente útil em determinados dias (feriados e finais de semana) e horários não comerciais, quando há um intervalo maior no tempo na circulação dos veículos.

Serviços de informação sobre transporte público podem ser implantados pelas prefeituras nas políticas de transporte urbano, fazendo uso de subsídios municipais para instalar displays informativos nos pontos de parada ou criando aplicativos para esse propósito e apresentando meios para estimular seu uso pela população. Nota-se que o mercado de redes veiculares possui avanços importantes em termos teóricos e soluções, porém há carência de sistemas implementados, até mesmo tratando-se de serviços básicos como as plataformas de localização e informação de ônibus do transporte urbano. Torna-se importante a implementação de protótipos e simuladores, principalmente em meio acadêmico, para avaliar a solução antes de colocá-la disponivel em prática.

O objetivo principal desse trabalho é propor um serviço que utiliza unicamente o sinal de internet *Wi-Fi* para detectar a aproximação entre pontos conectados à rede, de forma a determinar sobre qual ponto o veículo está mais próximo em um momento qualquer da trajetória e inserir a implementação em um ambiente reduzido/simulado,

de modo a testar e melhorar a interação entre as entidades envolvidas. Em complementação, busca-se implementar uma solução de baixo custo, desenvolver um meio alternativo de localização, promover a eficiência do transporte público, indicar melhorias para o tráfego urbano e colaborar com projetos acadêmicos sobre redes veiculares.

Para realizar este trabalho, foi seguida a seguinte metodologia:

a) estudo das técnicas para calcular a distância entre objetos apoiadas por comunicação via sinais *Wi-Fi*;
b) estudo da plataforma Arduino;
c) instalação e configuração do ambiente de programação;
d) execução do projeto do algoritmo;
e) implementação do algoritmo para calcular a distância entre objetos, compatível com as limitações do ambiente simulado;
f) implementação da solução usando a plataforma Arduino;
g) estudo das tecnologias para troca de informação na rede entre os pontos;
h) construção do cenário de teste;
i) execução de testes no ambiente controlado e avaliação da implementação.

O presente trabalho está estruturado do seguinte modo: o capítulo 2 contém uma revisão bibliográfica sobre os sistemas de auxílio ao usuário de transporte público urbano, apresentando os principais aplicativos existentes, mostrando suas principais características de uso. Em seguida há uma descrição dos trabalhos acadêmicos relacionados, destacando as semelhanças básicas de implementação em relação a este trabalho.

No capítulo 3 são apresentados todos os componentes presentes na implementação. O desenvolvimento do sistema proposto, tratando características da arquitetura envolvida, algoritmos e estratégias de implementação são tratados no capítulo 4. O uso do sistema de forma simulada, as configurações e a forma de funcionamento dos componentes envolvidos são descritos no capítulo 5.

Por fim, o capítulo 6 apresenta a conclusão deste trabalho, envolvendo contribuições a respeito do desenvolvimento do mesmo, possibilidades e desafios para implementá-lo em um ambiente real, listando também os possíveis trabalhos futuros para o melhoramento do sistema proposto.

2 REVISÃO DO ESTADO DA ARTE

Este capítulo está dividido em três partes: a primeira abrange os sistemas de transportes inteligentes e suas categorias, apresentando a colaboração deste trabalho em relação ao tema; a segunda parte apresenta brevemente o contexto atual em grandes cidades, listando os principais sistemas e aplicativos para auxílio ao usuário do transporte público; a última seção contém os principais trabalhos relacionados ao tema, comentando suas principais caracteristicas.

2.1 SISTEMAS DE TRANSPORTE INTELIGENTES - ITS

A partir do contínuo crescimento dos sistemas de trânsito, que envolve as vias públicas e também a frota de veículos que circulam, nota-se a necessidade de organização e otimização dos serviços prestados. É nesse contexto que surgem os ITS (*Intelligent Transportation Systems*), formando um conjunto de soluções tecnológicas avançadas que operam em conjunto com metodologias de captação de dados, com o intuito de proporcionar conforto, segurança e economia aos usuários e empresas envolvidas. Os ITS podem ser categorizados em (SUSSMAN, 2000):

a) Sistemas Avançados de Transporte Público (*Advanced Public Transportation Systems* ou APTS): sistemas que têm como objetivo melhorar a segurança e a efetividade do transporte público. Os benefícios que APTS oferecem aos usuários incluem redução dos tempos de espera, segurança e facilidade para o pagamento da tarifa, bem como provimento de informações precisas e atualizadas sobre itinerários e horários das linhas de ônibus, trens e metrô;
b) Sistemas Avançados de Gerenciamento de Tráfego (*Advanced Transportation Management Systems* ou ATMS): são os sistemas que buscam reduzir congestionamentos nas vias urbanas e rurais, garantindo segurança por meio de controle e monitoramento de semáforos e uso de câmeras de vídeo;
c) Sistemas Avançados de Informação ao Viajante (*Advanced Traveler Information Systems* ou ATIS): sistemas oferecem informações ao viajante sobre a via, e condições ambientais e de trânsito. São suportados por tecnologias como navegação automotiva e provimento de informação para garantir segurança ao motorista e para minimizar os congestionamentos;
d) Sistemas de Operação de Veículos Comerciais (*Commercial Vehicle Operations* ou CVO): sistemas que são usados para gerenciar a operação de veículos comerciais. Usam tecnologias para melhorar a gerência e o serviço

de transporte de cargas procurando minimizar interferências com relação às rotas e tempos perdidos, e assegurar alto nível de segurança.

e) Sistemas Avançados de Controle Veicular (*Advanced Vehicle Control Systems* ou AVCS): sistemas que buscam melhorar a segurança viária, permitindo que veículos auxiliem motoristas. Veículos são equipados com tecnologias que possibilitam que o motorista monitore as condições de dirigibilidade e tome ações no sentido de evitar acidentes.

f) Gestão Eletrônica de Tráfego e Pedágio (*Electronic Toll and Traffic Management* ou ETTM): constituem os sistemas que oferecem soluções eficientes para cobrança de pedágio, procurando reduzir o tempo necessário no processo de cobrança, e possivelmente, colaborar na redução de congestionamentos.

Todos os sistemas listados utilizam técnicas de localização para, em conjunto com outros parâmetros, efetuar cálculos e apresentar decisões. O sistema aqui implementado torna-se útil para os Sistemas Avançados de Transporte Público (APTS), ao apresentar as informações sobre o contexto e da mesma forma sobre os Sistemas Avançados de Gerenciamento de Tráfego (ATMS) e Sistemas Avançados de Informação ao Viajante (ATIS), usando as informações sobre aproximação para alertar sobre pontos de interesse (semáforos, trechos de congestionamento, entre outos). Os Sistemas de Operação de Veículos Comerciais (CVO) também podem se beneficiar dessa implementação, já que cálculos são feitos para otimização de rotas (necessita informações sobre localização) e também sobre posicionamento de veículos em mapas.

2.2 CONTEXTO ATUAL

Sistemas de aviso de aproximação entre veículo e ponto estático possuem diversas utilidades, dentre elas destaca-se o uso como meio de informação a passageiros de transporte coletivo urbano, alarmes para motoristas sobre finalização de um percurso, detecção de aproximação de um carro e um cruzamento, entre outros. Mais especificamente, no contexto de provimento para informação sobre transporte coletivo a usuários, existem sistemas que informam o horário de chegada do ônibus ao ponto de parada. No Brasil, destaca-se os serviços oferecidos as cidades Goiânia, Curitiba, São Paulo e Rio de Janeiro. Estes sistemas informacionais são baseados em algoritmos de previsão, e apresentam aos usuários informações confiáveis, quase em tempo real, sobre o tempo de espera e outras variáveis.

Cidades como Goiânia e Belo Horizonte possuem um serviço de aviso apresentado em *displays* nos pontos de parada, onde o usuário tem acesso visual direto sobre quanto tempo restante para que o próximo ônibus de cada linha local chegue.

Outras cidades realizaram aprimoramentos, criando plataformas capazes de manter uma linha de comunicação entre os ônibus e locais e parada, como por exemplo na cidade de São Carlos - SP, foi implantado uma ferramenta de apoio, o Busalert (BUSALERT, 2016). O sistema é um aplicativo para dispositivos móveis (celular e tablet) desenvolvido para auxiliar o passageiro, inclusive deficiente físico, deficiente visual, idoso, gestante entre outros, a monitorar as distâncias e/ou o tempo de chegada entre o ônibus mais próximo e o ponto de ônibus onde esse se encontra. O aplicativo conta com comunicação de áudio e texto, e assim o motorista pode ser informado antecipadamente se há deficientes físicos e/ou visuais em determinado ponto e esse aviso é dado pelo próprio passageiro enquanto aguarda, ou seja, há uma interação facilitada entre ambas as partes, evitando que o deficiente perca o ônibus por falta de autonomia e tornando o serviço prestado pelas empresas de transporte mais eficiente.

Em grandes cidades, onde geralmente o sistema de transporte é amplo (como em Londres, Paris, Berlim e Nova York), a sinalização e aplicativos para auxílio facilitam a locomoção dos visitantes. Em Londres, o aplicativo Citymapper (CITYMAPPER, 2016) apresenta o horário em tempo real da chegada de ônibus e metrô, mostra dicas de qual linha tomar para ir de um lugar a outro, oferece mapas e calcula tempos de percurso. Também apresenta facilidades para quem deseja usar a bicicleta, mostrando qual estação de bicicleta mais próxima, e qual melhor rota.

Várias outras cidades brasileiras possuem sistemas e aplicativos similares, que realizam consultas em um banco de dados (que geralmente fica sob responsabilidade dos órgãos de transporte público locais) conforme orientações apresentadas pelo usuário (linha de transporte desejada, localização e hora) e informam itinerários, pontos e horários de partida e chegada, tempo médio de percursos, etc. Alguns possuem interação com sistema de GPS para aprimorar o uso. São exemplos destes aplicativos:

a) CittaMobi (CITTAMOBI, 2015): Abrange atualmente 20 cidades brasileiras, como Salvador, São Paulo e Recife. Apresenta o horário de cada linha, usando informações do banco de dados contendo informações sobre a localização dos pontos de ônibus, integrado ao GPS. O aplicativo permite também a verificação do tempo aproximado de chegada dos veículos em cada ponto, qual o ponto mais próximo da localização atual e informa se o veículo que está a caminho é adaptado para portadores de deficiência física. Disponível em sistemas web, painéis para terminais e Aplicativo móvel (Android e iOS);
b) Moovit (MOOVIT, 2016): Segundo o site do aplicativo, existem 28 milhões de usuários, e o sistema está disponível em 58 países, 700 cidades (47 brasileiras). Apresenta informações para ônibus, trem e metrô. Utiliza como

apoio o sistema de GPS para localização básica do usuário e permite que sejam informados dados de origem e destino, apresentando informações sobre a distância do trajeto e o tempo aproximado para percorrê-lo. O aplicativo conta também com uma comunidade para compartilhamento de informações, onde é possível por exemplo qualificar o desempenho de motoristas e apontar problemas apresentados na locomoção. Disponível para as plataformas Android, iOS e Windows Phone;

c) Trafi (TRAFI, 2016): Também disponível para vários meios de transporte, opera em 8 países e está há pouco tempo no Brasil (apenas para as cidades do Rio de Janeiro e São Paulo) e por isso está em fase de correção de problemas. A empresa responsável pelo aplicativo promete trazer novos algoritmos (incluindo algoritmos de auto-aprendizagem) para melhorar o processamento em tempo real, prevendo comportamento do tráfego através do estudo dos padrões locais (para auxílio no cálculo de tempos) sem necessidade de inserção manual de dados. Apresenta informações bastante peculiares, como por exemplo, se o ônibus está atrasado ou se está lotado. Um destaque é a integração com os usuários, que permite ao sistema aplicar filtros para apresentar notícias importantes conforme perfis individuais. Outro diferencial é a possibilidade de trabalhar *off-line*, mesmo ocupando pouco espaço de armazenamento. Disponível para as plataformas Android, iOS e versão para Web;

d) SIU Mobile (TACOM, 2016): Disponível para as cidades de Belo Horizonte, Terezina, Salvador e Porto Alegre, opera em sistemas Android, iOS e Windows Phone. Além de todas as funcionalidades básicas de localização, apresenta apoio a portadores de deficiência visual, disponibilizando funcionalidades para adaptar o layout do aplicativo ao modo de navegação do usuário. É possível também identificar os veículos adaptados para pessoas com mobilidade reduzida. Os pontos de parada são cadastrados no banco de dados do aplicativo, de modo a garantir a perfeita localização e integração com o GPS.

Além do serviço comum prestado, nota-se que os aplicativos Busalert e SIU Mobile possuem especialidades, como por exemplo, o atendimento a pessoas portadoras de necessidades especiais e outros como Trafi e Citymapper possuem atendimento a vários meios de transportes. Existem também vários aplicativos pequenos, gerados por meio de empresas ou órgãos responsáveis pelo transporte público, que operam de forma local em diversas cidades, como é o caso do Lisboa MOVE ME (OPT, 2016), exclusivo para a cidade de Lisboa - Portugal, que não opera em tempo real e usa apenas informações de banco de dados para informar horários e estimar os tempos de

chegada dos veículos.

Muitos sistemas que trabalham com informações em tempo real fazem uso do sistema de GPS, o que pode envolver transmissão de grande quantidade de dados, muitas vezes desnecessária para apresentar uma simples informação em um *display* ou aplicativo, sobre, por exemplo, qual próximo ônibus chegará ao terminal. O uso de uma rede *Wi-Fi*, com trocas de sinais de aviso entre veículos e pontos de parada seria uma opção viável, para tornar ferramentas mais robustas, sem compartilhamento desnecessário de informações.

2.3 TRABALHOS ACADÊMICOS RELACIONADOS

A plataforma Arduino e seus componentes de conexão *Wi-Fi* tem sido frequentemente utilizada para prototipar serviços como detecção de presença em ambientes, localização de objetos e para construir bases para mapeamento. Também há uma série de trabalhos que utilizam redes *Wi-Fi* em técnicas para obtenção de informações de localização no tráfego, mas não foram encontrados estudos que combinem sistemas Arduino, trânsito e redes sem fio.

Como exemplos de trabalho que utilizam localização por *Wi-Fi*, destaca-se: (CONRAD, 2014) e (TAN; WONG, 2016). Em (CONRAD, 2014), a partir do fato de que sistemas de localização por *Wi-Fi* são mais precisos do que GPS em ambientes internos, os autores apresentam um aplicativo de segurança capaz de rastrear funcionários dentro de um edifício, de forma a facilitar a localização de uma saída de emergência quando alguém enviar um mensagem de aviso de incidente.

Em (TAN; WONG, 2016) é apresentado um serviço de baixo custo para posicionamento de veículo usando pontos de acesso (access points) *Wi-Fi*. O serviço proposto consiste em um sistema de rastreamento de ônibus para estimar o tempo de chegada nos pontos de parada, no campus da universidade de Malaya, utilizando técnicas de aproximação via *Wi-Fi* (com auxílio de GPS), envio de dados de forma manual pelo motorista e um *website* para apresentar as informações. Esse trabalho utiliza a rede *Wi-Fi* como apoio a localização em conjunto com pontos de acesso e tem semelhança com este trabalho de Graduação, apesar de não utilizar placas Arduino ou similares.

Existem trabalhos que apresentam sistemas alternativos diversos para localização precisa de veículos nas vias. Como exemplos podem ser citados (SAVVIDES; HAN; STRIVASTAVA, 2001) e (BOUKERCHE et al., 2008), que apesar de não serem recentes, possuem importância no contexto atual. Em (SAVVIDES; HAN; STRIVASTAVA, 2001) é tratado um método de localização fazendo uso exclusivo de sensores a partir de um sistema chamado AHLoS (*Ad-Hoc Localization System*) que permite que os próprios sensores descubram sua localização, usando um conjunto distribuído de

algoritmos iterativos. O estudo do protótipo é feito a partir de uma simulação, apresentando uma precisão com erro de poucos centímetros.

Em (BOUKERCHE et al., 2008) é relatado um estudo sobre diversas técnicas alternativas de localização, como por exemplo, localização por celular e localização por imagem e vídeo, e maneiras de como estas técnicas de localização podem ser combinadas usando técnicas de fusão de dados para fornecer a localização mais precisa possível. A análise leva em conta a expansão contínua das redes veiculares, e possíveis falhas no sinal de GPS em lugares críticos.

Outro ponto importante é a precisão na troca de sinais em redes ponto-a-ponto (*P2P*), como é tratado em (CHEN; KUNG; VLAH, 2001), que foca momentos de baixa densidade de tráfego, quando os pontos estão mais distantes e partes da rede podem ficar desconectadas. É apresentado um estudo sobre a retransmissão de dados, que pode ser útil nesses momentos, sendo necessário para isso o armazenamento temporário nos pontos, até que o movimento dos veículos torne favorável o repasse da informação.

Finalmente, como referencia informal, (BR-ARDUINO, 2015) apresenta uma forma de detectar informações (apresentadas em um *display* simples) sobre as redes Wi-Fi disponíveis em um dado ambiente, com auxílio do módulo ESP8266 conectado a uma placa Arduino UNO, de modo a identificar os canais de transmissão menos congestionados para uma melhor configuração do roteador. É importante ressaltar que existem inúmeros sites na Web, reportando implementações Arduino.

3 SERVIÇO DE INFORMAÇÃO DE PROXIMIDADE DE VEÍCULO E PONTOS DE INTERESSE

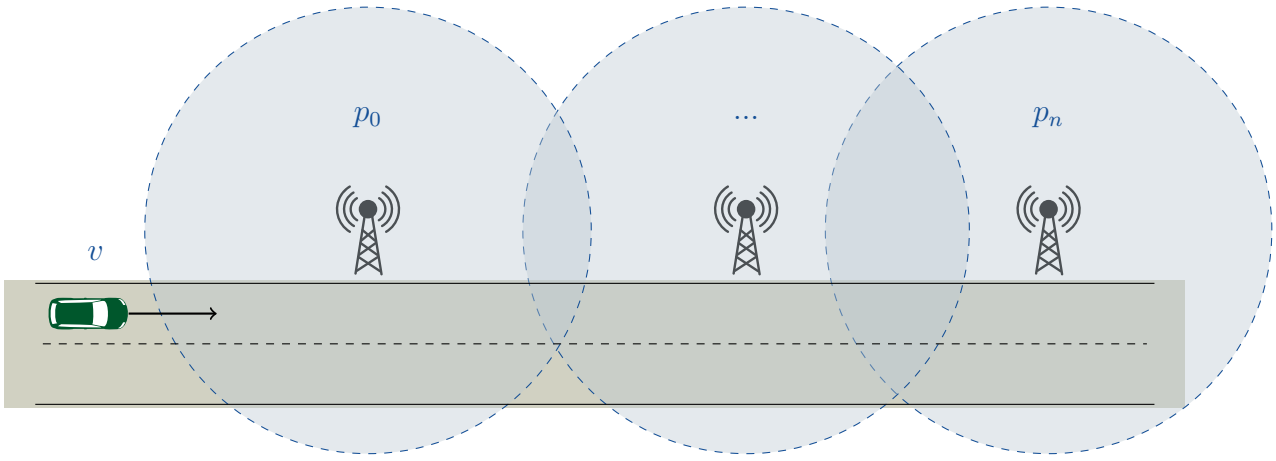
O presente capítulo apresenta a formulação do modelo teórico e descreve os componentes necessários na sua composição, explorando suas principais caracteristicas e de que forma influenciam na operação do conjunto.

3.1 MODELO TEÓRICO

O cenário do problema em questão é composto por um veículo que percorre um trajeto e pontos estáticos com localização conhecida. Tais pontos podem ser semáforos, cruzamentos, pontos de ônibus ou qualquer estrutura fixa de interesse. Um veículo v move-se em direção a $n+1$ pontos de interesse p_i. A jornada do veículo é conhecida: sabe-se a origem, o destino e a sequencia que os pontos de interesse devem ser atingidos. O veículo v sempre passa pelo ponto p_i depois de passar pelo ponto p_{i-1}.

A Figura 3.1 apresenta este cenário. Cada ponto p possui uma forma de conexão com internet *Wi-Fi*, contendo determinada abrangência e força de sinal. É possível saber sobre qual ponto o veículo está mais próximo no decorrer da trajetória, realizando constantemente a medição da força do sinal nos pontos e no veículo (como será melhor explicado nas seções seguintes) e dessa forma mapear o andamento, detectando entre quais pontos o veículo se encontra.

Figura 3.1 – O cenário alvo com uma via, $n+1$ pontos de interesse e um veículo v com uma jornada a percorrer



Fonte: Autor.

O veículo no cenário pode ser, por exemplo, um carro de passeio em relação a semáforos ou cruzamentos, uma ambulância em relação a um local de acidente, um caminhão em relação a pontos de carga/descarga, ou como está sendo abordado neste trabalho, um ônibus em relação a pontos de parada.

3.2 PLATAFORMA ARDUINO

Devido a sua alta flexibilidade, foi utilizado neste trabalho o sistema de desenvolvimento Arduino. O Arduino é uma plataforma de prototipagem eletrônica aberta que se baseia em *hardware* e *software* flexíveis e fáceis de usar. Existem diversos modelos disponíveis, alguns mais básicos, utilizados para gerenciar poucas funções em projetos simples, como por exemplo, controle de luminosidade ambiental, e outros mais aprimorados (maior poder de processamento, memória RAM, entradas para conexões, etc) que podem controlar vários processos e serem acoplados com outros módulos, como é o caso do Arduino 2560 utilizado no carro-robô deste projeto, que controla todos os sensores de movimento, acopla-se ao módulo que controla os motores (L298N *DC Dual H-Bridge*) e ainda acopla-se ao Módulo ESP8266, responsável pela comunicação *Wi-Fi*. Todos os modelos Arduino são programados da mesma forma, com a mesma base de programação (C/C++), fazendo-se uso do mesmo SDK (*Software Development Kit*).

De forma básica, uma placa Arduino possui processador, memória RAM, conexões digitais e analógicas de entrada e saída, conversor para processamento das entradas analógicas e uma porta serial para comunicação com o computador, que possibilita gravar e atualizar o programa fonte a ser executado na memória da placa, ou seja, há uma fácil interconexão entre *hardware* e *software*. O Arduino ainda pode receber auxílio de um computador em tempo de execução, conforme complexidade do projeto, utilizando para isso um *software* específico (como exemplo, *Flash* e *Processing*). Por ser uma plataforma aberta, as placas podem ser construídas pelo usuário, (o *hardware* pode ser adaptado conforme necessidade) ou podem ser adquiridas já montadas, conforme projeto original Arduino.

3.3 *HARDWARE* UTILIZADO

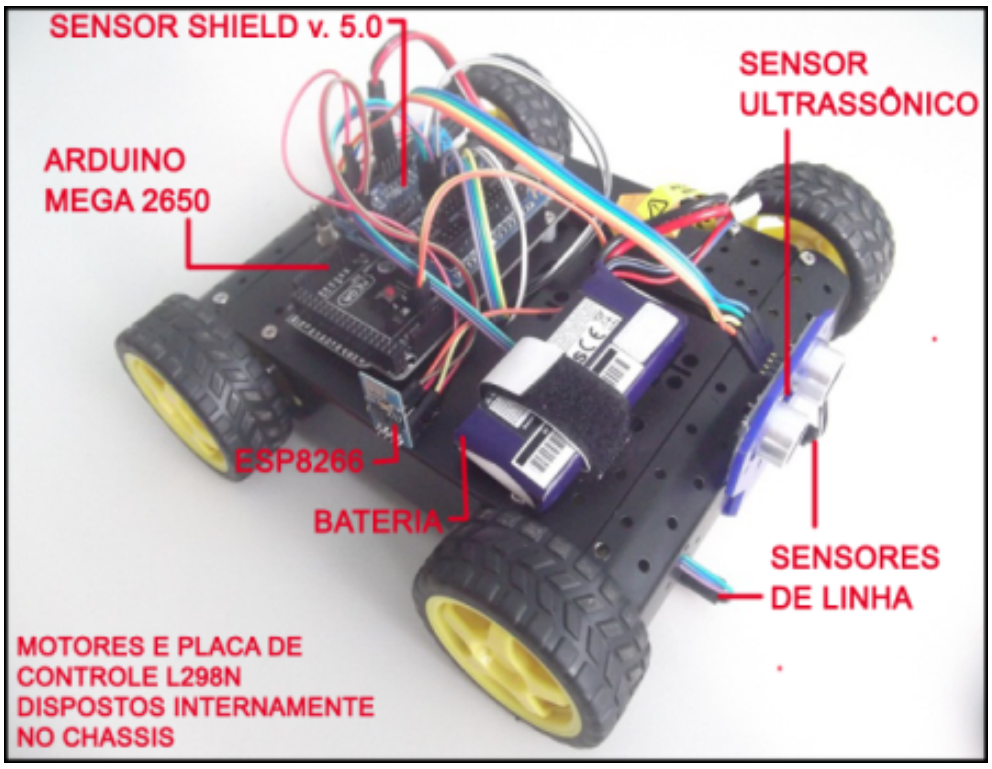
3.3.1 Veículo robô

A base do modelo é composta por um veículo robô, assistido por uma placa Arduino e módulos secundários e 4 computadores convencionais, sendo um deles usado como servidor, responsável pela execução de algoritmo principal, apresentação de resultados e coletagem de dados dos componentes restantes (3 computadores e o

Arduino) que são tratados como clientes. Os computadores clientes são usados para disparar dados a respeito dos pontos de parada no ambiente de simulação.

O veículo robô (Figura 3.2) é composto por um chassi com 4 rodas, uma placa Arduino Mega 2560, quatro motores para controlar as rodas de forma independente, uma placa para controlar os motores (L298N DC Dual H-Bridge), uma placa para conectar sensores e os outros componentes ao Arduino (sensor shield v. 5.0), uma bateria Li-Po de 1000 mAh, um sensor ultrasônico (HC-SR04), um par de sensores seguidores de linha (que permite o carro seguir sobre uma linha instalada no piso para representar o trajeto) e um módulo *Wi-Fi* (ESP8266) usado para a comunicação.

Figura 3.2 – Veículo robô utilizado



Fonte: Autor.

A placa Arduino Mega 2560 possui toda a base para apoiar seu microcontrolador ATmega2560 (que contem 54 pinos digitais de entrada). Para alimentação, basta conectá-lo a um computador com um cabo USB ou ligá-lo com um adaptador AC-DC ou bateria, como é o caso neste trabalho, já que o veículo controlado pela placa permanece em movimento. O 2560 mega é compatível com a maioria dos escudos projetados para o Arduino Uno e outros modelos mais antigos (ARDUINO, 2015). Os dados referentes a correntes de trabalho, capacidade de memórias e outros itens estão na Tabela 3.1.

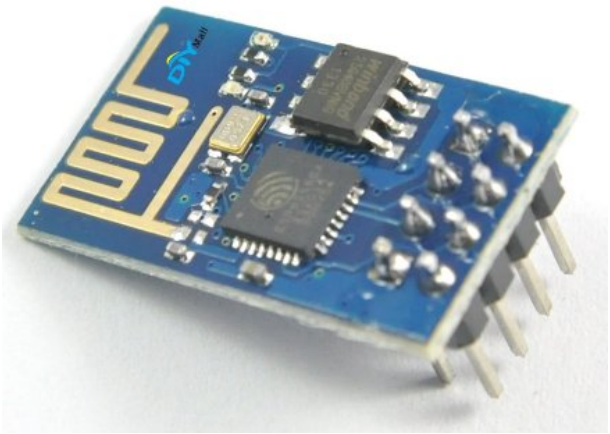
Tabela 3.1 – Caracteristicas do módulo Arduino 2560

Caracteristica	**Valor**
Tensão de funcionamento	5V
Tensão de entrada (recomendado)	7 a 12V
Tensão de entrada (máxima)	6 a 20V
Pinos de entrada e saida digital	54
Pinos de entradas analógicas	16
Valor máximo de corrente fornecida por pino	40mA
Valor de corrente para pino 3,3V	50mA
Memória *flash*	256KB
SRAM	8KB
EEPROM	4KB
Velocidade de *clock*	16MHz

Fonte: Adaptado de (ARDUINO.CC, 2015).

O módulo de comunicação *Wi-Fi* do veículo robô é o microcontrolador ESP8266, fabricado pela empresa chinesa Espressif (http://bbs.espressif.com), modelo ESP-01 (Figura 3.3). Ele possui sistema *Wi-Fi* embutido (*System-On-Chip*) para comunicação TCP/IP, sensor interno de temperatura, processador que opera em 80 MHz (com possibilidade de operar em 160 MHz) e arquitetura RISC de 32 bits. Tratando de memórias, possui 32 Kbytes de RAM para instruções, 96 Kbytes de RAM para dados, 64 Kbytes de ROM para boot e memória *Flash* de 512 Kbytes. O fabricante mantém em seu site, um fórum para troca de informações e um repositório para documentações.

Figura 3.3 – Módulo para comunicação *Wi-Fi* - ESP8266



Fonte: Sourceforge (2015).

O módulo L298N contém um circuito *H-Bridge* que pode conduzir uma corrente em qualquer polaridade e é normalmente utilizado no controle de velocidade e direção de motores, mas pode ser usado para outros projetos, tais como controle de brilho de projetos de iluminação, como matrizes de *LED* de alta potência (INSTRUCTABLES, 2014). O L298N e os outros componentes citados são acoplados ao arduino através do *Sensor Shield* v5.0, que tem a capacidade de expandir as entradas e saídas (digitais e analógicas) do Arduino, tornando possível a instalação de módulos extras (chamados servos). Ele possui uma entrada individual de energia para as conexões, onde um jumper permite ao usuário selecionar o uso de energia interna ou externa, para controlar os servos (SHIELDS-MANUAL, 2010).

3.3.2 Pontos estáticos

Foram utilizados nos pontos estáticos computadores de modelos de baixo custo, em conjunto com máquinas mais aprimoradas. Dois computadores utilizados são da marca CCE *Win*, processador Intel Core I3 (1.9GHz), possuindo 3Gb de memória RAM, e em outro ponto estático ficou em operação um MacBook Pro, processador Intel Core I5 (2,7GHz) com 8Gb de memória RAM. No caso do computador responsável pela operação do cliente, temos um *desktop* Mac Mini, com processador Intel Core I5 (2,3GHz) e 8GB de memória RAM.

3.4 SOFTWARE UTILIZADO

A plataforma Arduino conta com um IDE de programação própria e neste trabalho foi utilizada a versão 1.6.12. A linguagem Arduino é baseada em C / C ++, e o processamento de linguagem, é baseado em Java (ARDUINO-REFERENCE, 2015). Além das bibliotecas default da IDE, foram utilizadas as bibliotecas “ESP8266.h”, que contém funções responsáveis pelo controle e comunicação *Wi-Fi* do Módulo ESP8266 com o notebook servidor e a biblioteca “NewPing.h”’, que proporciona o controle dos sensores ultrassônicos dianteiros do veículo-robô, responsáveis por detectar a presença de um obstáculo a uma distância determinada, impedindo a colisão.

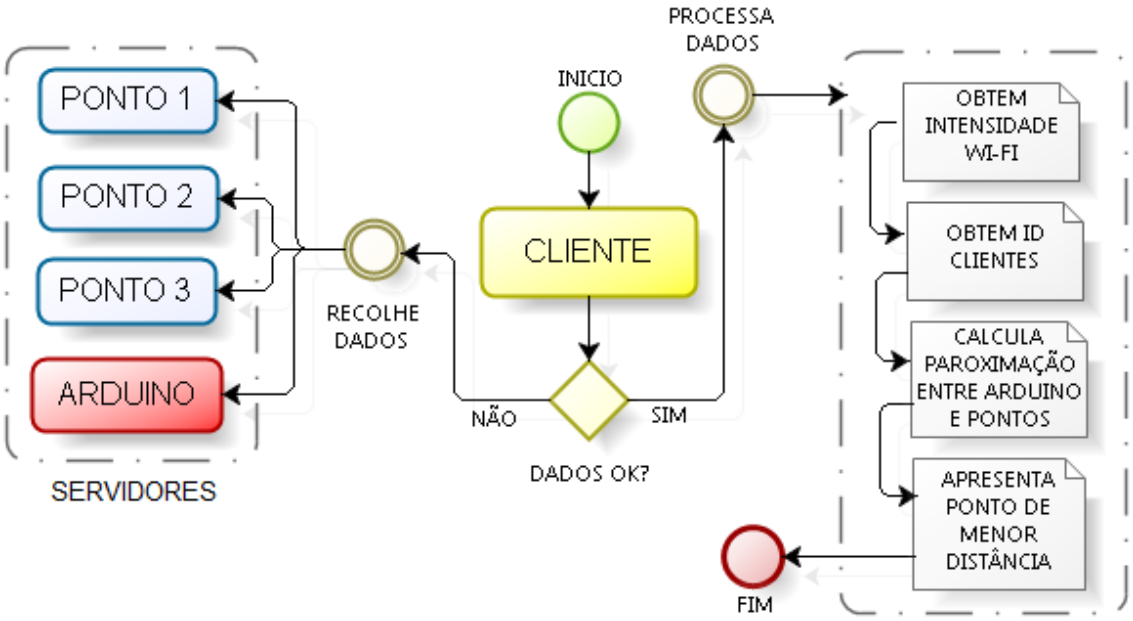
A programação nos computadores convencionais (Clientes e Servidor) é feita na linguagem Java, e a comunicação Clientes/Servidor é realizada através de Sockets. O IDE utilizado para o desenvolvimento foi o NetBeans versão 8.1, em conjunto com o pacote de desenvolvimento Java (JDK) versão 1.8, sem bibliotecas adicionais. Não houve problemas de comunicação entre Sockets Java e o ESP8266.

O sistema operacional instalado nos computadores servidores CCE Win, foi o Linux Ubuntu versão 16.04 LTS. O servidor Mac operou com o *OS X Yosemite* versão 10.10, enquanto que o *desktop* cliente utilizou o *OS X Lion* versão 10.7.

4 IMPLEMENTAÇÃO

A arquitetura da implementação é composta por um veículo-robô (assistido por uma placa Arduino com módulos secundários) e 4 computadores convencionais, sendo um deles usado como cliente, responsável pela conexão principal, apresentação de resultados e coletagem de dados dos componentes restantes (3 computadores e o Arduino) que são tratados como servidores, como é possível ver na Figura 4.1. Os computadores servidores são usados para enviar dados a respeito dos pontos de estáticos, onde estarão localizados no ambiente de simulação. Cada servidor deve repasar ao cliente a intensidade do sinal *Wi-Fi* em sua localização (medida em decibéis por minuto - DBm). Com esse dado, o servidor tem a capacidade de identificar sobre qual ponto de parada (*id*) o veículo está mais próximo. Os códigos de todos os componentes constam nos Apêndices.

Figura 4.1 – Arquitetura da implementação



Fonte: Autor.

4.1 CAPTURA DA INTENSIDADE DO SINAL *WI-FI*

Na implementação, o objetivo principal é a obtenção da informação sobre a intensidade do sinal *Wi-Fi* nos componentes. Dessa forma, iniciou-se um estudo para determinar a melhor forma para detectar e armazenar essa informação como um dado no algoritmo, tanto no Arduino (veículo) como nos computadores (pontos de parada).

4.1.1 Pontos de parada - computadores convencionais

A plataforma Linux possui arquivos virtuais contendo informações sobre o sistema, que podem ser acessados facilmente por um programa. Um dos diretórios desse sistema de arquivos é o “/proc”, ele não contém arquivos reais, mas as informações do sistema em tempo de execução (por exemplo, sistema de memória, dispositivos montados, configuração de hardware, etc). Por esta razão, pode ser considerado como um centro de controle e informação para o kernel (LINUX, 2001). O arquivo de interesse para a implementação localiza-se em ”/proc/net/wireless” e apresenta seu conteúdo (Figura 4.2) no formato de texto simples (.txt). Os dados *quality-link, quality-level* e *quality-noise* são referentes a qualidade do sinal de internet *Wi-Fi*, onde o primeiro (*quality-link*) apresenta a qualidade geral da recepção (levando em consideração pacotes perdidos), medido numa escala de 0 a 100 (pode variar conforme o sistema). O segundo (*quality-level*) apresenta a força do sinal no receptor, medido em DBm (Decibéis por minuto), em uma escala negativa que varia de 0 a -100 (quanto mais próximo de 0, melhor é o sinal) e o terceiro valor (*quality-noise*), representa o nível de ruído (analizando periódos sem pacotes no receptor), em escala percentual. Quando o parâmetro está indisponível no sistema, é apresentado o valor -256.

Figura 4.2 – Conteúdo do arquivo virtual "wireless"(Linux Ubuntu 16.04)

```
Inter-| sta-|   Quality        |   Discarded packets               | Missed | WE
 face | tus | link level noise |  nwid  crypt   frag  retry   misc | beacon | 22
wlp6s0: 0000   65.  -45.  -256        0      0      0    578   1299        0
```

Fonte: Autor.

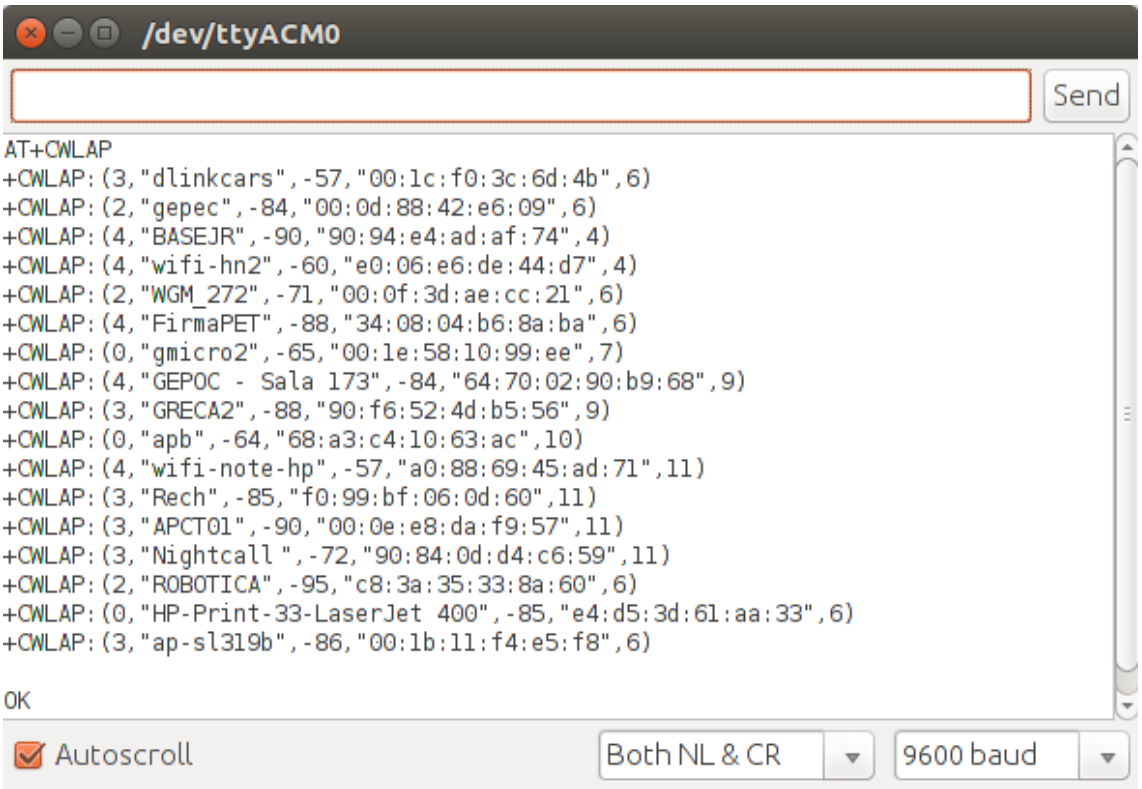
Com acesso ao conteúdo desse arquivo, o algoritmo busca o conteúdo do parâmetro *quality-level* na terceira linha, sempre que for necessário enviar o dado para o computador de controle (em caso de requisição).

O procedimento para obtenção das informações da rede no caso da plataforma MacOS ocorre por linha de comando, obtendo-se o conteúdo apresentado por um processo do sistema que contém as informações de todas as redes ao alcance. Esse conteúdo é então armazenado em uma *string* e dessa forma torna-se possível encontrar a rede a qual o notebook encontra-se conectado e buscar o dado referente a intensidade do sinal *Wi-Fi*.

4.1.2 Veículo - Arduino e módulo ESP8266

Tratando-se do ESP8266, uma forma de realizar controles e obter dados é através dos comandos AT (quando conectado ao Arduino). Comandos AT estão presentes nas linguagens de comando de diversos equipamentos periféricos de computadores, e são utilizadas para programação de *drivers* e *firmwares* em geral (como os módulos ESP). O comando AT base para a obtenção da intensidade sinal *Wi-Fi* denomina-se “AT+CWLAP”, que retorna uma lista com todos os *Access Points* que estão sendo detectados pelo ESP (Figura 4.3).

Figura 4.3 – Resposta do comando ”AT+CWLAP” na plataforma Arduino



Fonte: Autor.

Cada linha é composta por 5 elementos informativos, que são:

a) segurança: o tipo de criptografia da rede, sendo 1 para WEP, 2 para WPA e 3 para WPA2. O valor 0 indica que a rede não é criptografada;
b) SSID (*Service Set Identifier*): O nome da rede;
c) Potência: O nível do sinal, em DBm (dado buscado);
d) MAC (*Media Access Control*): O endereço de rede físico do equipamento;
e) o número do canal adotado por aquela rede;

Ao ter disponível o SSID da rede, o endereço físico, e o canal utilizado por ela, é possível reduzir a quantidade de dados gerados e otimizar a busca pela informação referente a rede conectada, passando tais informações como parâmetros ao comando “AT+CWLAP”. Por exemplo, se os componentes estão conectados a uma rede chamada “dlinkcars”, que possui endereço físico “00:1c:f0:3d:6d:4b” e opera no canal número seis, basta realizar o comando “AT+CWLAP="dlinkcars","00:1c:f0:3d:6d:4b",6”. O retorno dentro do algoritmo é uma string que contém 3 caracteres, onde o primeiro é o sinal negativo (descartado posteriormente) e os outros dois caracteres referem-se a intensidade do sinal na escala de medida. Apesar da escala ir até -100, o sinal não chegará a esse valor, porque devido a obstáculos na propagação e interferências constantes, a conexão é perdida antes que o pior nível possa ser detectado.

4.2 REPASSE DE DADOS

4.2.1 Programação nos servidores

O módulo ESP8266 é controlado pelo módulo Arduino e envia a informação de sinal de modo constante pela sua porta de comunicação, para evitar um novo reconhecimento da rede sempre que houver uma requisição. Já os computadores convencionais dos pontos fixos realizam o envio do valor do sinal apenas quando requisitado pelo cliente, já que o sistema operacional possui a rede reconhecida e mantém tal informação sempre atualizada por processos internos.

O módulo Arduino possui uma função de configuração (*setup*) que é executada sempre que a placa é energizada. Nela ocorre a conexão do módulo com a rede, a habilitação de funcionalidades (como por exemplo, possibilitar múltiplas conexões) e a criação do socket na saída de comunicação *Wi-Fi* (Serial1), configurado na porta reservada para o veículo (8090). Na sequência um *loop* principal é executado, onde ocorre a captação do sinal *Wi-Fi* com o auxílio do módulo ESP8266 e após é realizado o envio da informação através do socket.

Nos computadores, o servidor é aberto em sua porta de comunicação e assim permanece aguardando requisição do cliente. Quando isso ocorre, uma *thread* é chamada, onde ocorre a criação do objeto de comunicação, a obtenção e envio da intensidade do sinal *Wi-Fi* e o fechamento da comunicação. Após isso o servidor segue em execução, aguardando a próxima requisição.

4.2.2 Programação no cliente

O computador responsável pelo recolhimento de todos os dados deve estar conectado à mesma rede dos servidores (via cabo ou *Wi-Fi*). Este componente opera

como um cliente que executa somente quando ocorre uma requisição de posicionamento do veículo pelo usuário. O Algoritmo do cliente deve ser informado (de forma ordenada) sobre os endereços de *IP* na rede, de todos os componentes e também sobre as portas de comunicação, enquanto que os servidores precisam ter a informação apenas sobre sua porta de comunicação com o cliente.

O processo inicial do algoritmo consiste em gerar sockets para cada *IP* de servidor registrado e realizar a conexão, que ocorre de forma ordenada (seguindo ordem de armazenamento de *IP*s): A primeira conexão é sempre com o Arduino, a segunda conexão é com o primeiro ponto de parada, a terceira conexão é com o segundo ponto de parada, e assim sucessivamente. O algoritmo apresenta as informações de conexão, conforme o andamento do processo (Figura 4.4). Caso ocorra uma falha em alguma conexão, o cliente informa a ocorrência e pára a execução.

Figura 4.4 – Informações das conexões entre os servidores e cliente

```
Conectando ao Socket número 1....
socket 1 conectado! (IP: 192.168.0.112; PORTA: 8090)
Conectando ao Socket número 2....
socket 2 conectado! (IP: 192.168.0.111; PORTA: 8091)
Conectando ao Socket número 3....
socket 3 conectado! (IP: 192.168.0.110; PORTA: 8092)
Conectando ao Socket número 4....
socket 4 conectado! (IP: 192.168.0.108; PORTA: 8093)
Todos os sockets conectados com sucesso!
```

Fonte: Autor.

Após realizadas todas as conexões, o programa realiza a próxima etapa, que consiste na confirmação de conexão para cada *IP* e a obtenção do dado (intensidade do sinal *Wi-Fi*) para cada conexão confirmada (Figura 4.5). A identificação do *id* do ponto de parada é feita diretamente conforme a ordem de endereços *IP* armazenados em um *array* no algoritmo, o recebimento e armazenagem dos dados (em um vetor de inteiros) também é feito de forma ordenada, do mesmo modo que ocorrem as conexões, ou seja, sabe-se que o dado armazenado na primeira posição é referente ao Arduino, na segunda posição é referente ao primeiro ponto de parada, na terceira posição é referente ao segundo ponto de parada e assim sucessivamente.

No momento em que os dados de todos os servidores foram recebidos com sucesso, ocorre a definição de posicionamento do veículo, leitura e atualização do mapa, conforme apresentado na próxima seção.

Figura 4.5 – Confirmação das conexões e recebimento dos dados no cliente

```
Arduino conectado. Dado Recebido: 77
Ponto de parada nº1 conectado. Dado Recebido: 64
Ponto de parada nº2 conectado. Dado Recebido: 63
Ponto de parada nº3 conectado. Dado Recebido: 26
```

Fonte: Autor.

4.3 MAPEAMENTO

Para a aplicação que este trabalho propõe (linha de ônibus em um percurso de rotina), o mapeamento do trajeto é feito de forma fixa, tomando como referência os pontos de parada para determinar a posição do veículo. Os pontos são numerados e dispostos em sequência na trajetória, ou seja, além da possibilidade de identificar a posição do veículo em relação aos próprios pontos, é possível também medir aproximadamente quanto do trajeto foi percorrido e quando falta para que o veículo chegue ao ponto final.

Durante o trajeto do veículo na implementação, a proximidade máxima em relação a um ponto de parada é detectada quando a diferença na intensidade do sinal *Wi-Fi* entre as entidades é menor que 8 DBm. Não é possível usar um valor mais baixo devido à instabilidade constante dessa métrica, que é afetada por ruídos e interferências do meio e pelo fato de que a antena de conexão do veículo com a rede jamais ocupará exatamente a mesma posição da antena de conexão do ponto de parada.

Para realizar o mapeamento, foi utilizado um arquivo de texto (denominado “map.txt”), onde cada linha representa um ponto de parada e pode conter apenas dois valores:

a) 0 (zero): o veículo não passou pelo ponto;
b) 1 (um): o veículo passou pelo ponto, ou está muito próximo;

A primeira linha do arquivo representa o ponto número 1, a segunda linha representa o ponto número 2 e assim sucessivamente. Levando em conta o trajeto entre o ponto de partida do veículo e o ponto de chegada, podemos ter três situações de mapeamento:

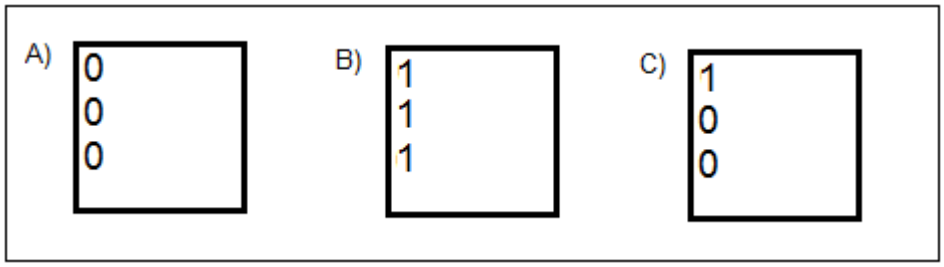
a) o veículo não passou por nenhum ponto estático;
b) o veículo já passou por todos os pontos estáticos;
c) o veículo está entre dois pontos estáticos quaisquer.

Ao iniciar a simulação, todas as linhas possuem o valor 0 (zero), representado um estado onde o veículo não passou por ponto algum. Nesta simulação está sendo

usado 3 pontos estáticos, portanto o arquivo inicial possui três linhas contendo o valor 0 (zero), e uma última linha vazia representando o fim do arquivo como mostra a Figura 4.6 (a).

No estado em que o veículo passou por todos os pontos, todas as linhas ficam armazenando o valor 1 (um), como apresentado na Figura 4.6 (b). Ao final da simulação, quando uma leitura de posição detectar que o veículo passou pelo último ponto, o arquivo é reescrito automaticamente com todas as linhas contendo o valor 0 (zero), e uma mensagem é apresentada, informando que o trajeto pode ser reiniciado.

Figura 4.6 – Exemplos de resentações do arquivo ”map.txt” durante a simulação



Fonte: Autor.

O último estado possível se apresenta quando o veículo está no decorrer do trajeto, ou seja, passou pelo primeiro ponto mas não passou pelo último ponto. Como exemplo, há o caso onde o veículo passou apenas pelo primeiro ponto na trajetória. Quando a leitura do mapa é feita nesse estado, o algoritmo detecta a primeira linha do arquivo “map.txt” que contém o valor 0 (zero) e dessa forma pode detectar entre quais pontos o veículo se encontra. Na leitura do mapa da Figura 4.6 (c), a segunda linha é detectada e dessa forma sabe-se que o veículo encontra-se antes do ponto número 2 e após o número 1 .

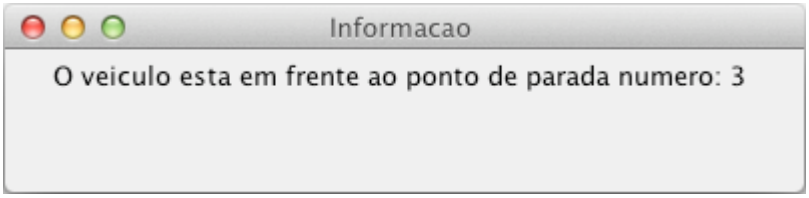
A alteração do mapa é feita sempre que o veículo detecta sua máxima aproximação em relação a um ponto estático (diferença de sinal *Wi-Fi* menor que 8 DBm). Neste momento, a primeira linha do arquivo que contém o valor 0 (zero) é detectada para decidir sobre qual ponto de parada o veículo se encontra próximo. Tal linha tem seu conteúdo alterado para o valor 1 (um), levando em conta que os pontos estão em sequência.

4.4 APRESENTAÇÃO DE LOCALIZAÇÃO

Com a finalidade de facilitar a visualização do resultado de cada execução, o programa do cliente apresenta uma janela (Figura 4.7) que possui a informação

a respeito da localização do veículo, além de também fazer a exibição no terminal de saída do programa, logo após as informações de conexão e exibição dos dados recebidos. A apresentação da janela sinaliza que todas as etapas do programa foram executadas com sucesso e no momento em que é fechada, o programa é finalizado, já com a atualização do mapa realizada, se necessário.

Figura 4.7 – Exemplo de janela informativa



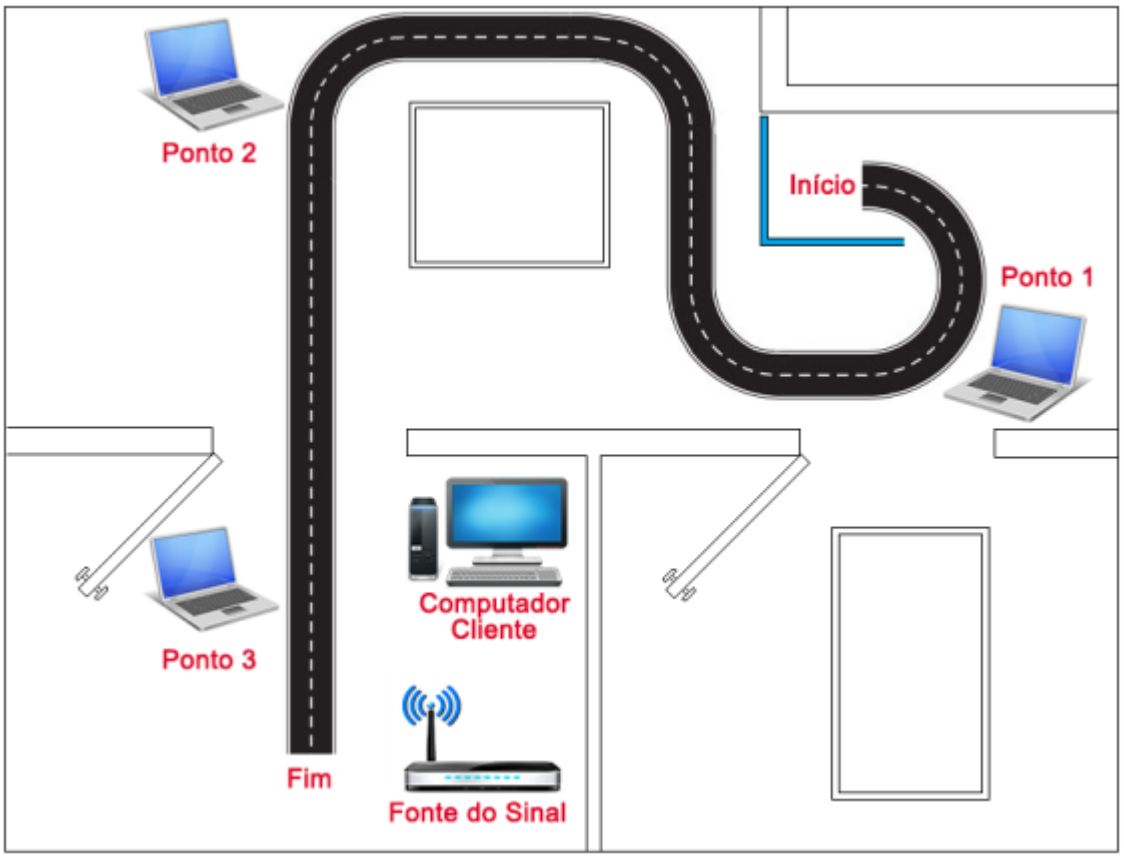
Fonte: Autor.

5 AVALIAÇÃO EXPERIMENTAL

5.1 AMBIENTE DE SIMULAÇÃO

A execução da simulação foi realizada no Laboratório de Sistemas de computação, no Centro de tecnologia da UFSM. O ambiente mostrou-se propício para a aplicação, já que havia acesso a internet *Wi-Fi*, um conjunto de máquinas disponíveis para os testes e diversas possibilidades de posicionamento dos pontos de parada (incluindo a presença de obstáculos) que devem ficar significamente separados para representar da melhor forma o ambiente real e também para que exista a diferença na intensidade do sinal *Wi-Fi* entre eles. O modelo do cenário pode ser visualizado na Figura 5.1, que apresenta os obstáculos no cenário, (paredes, mesas, portas, salas, etc) e o posicionamento dos pontos de parada no decorrer do trajeto escolhido para o veículo.

Figura 5.1 – Cenário de testes



Fonte: Autor.

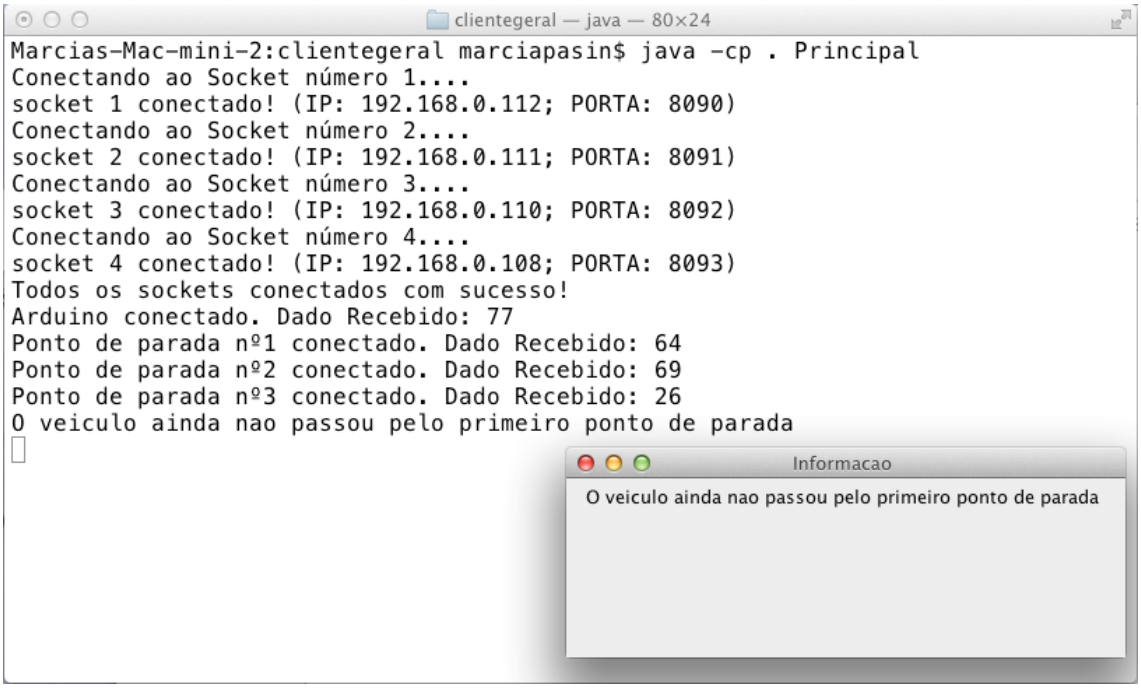
O ponto inicial da trajetória encontra-se atrás de um obstáculo para garantir

assim a simulação de distanciamento e diferença de sinal em relação ao primeiro ponto de parada, já que o ambiente limitou essa possibilidade. Com o mesmo objetivo, o último ponto estático encontra-se dentro de uma sala, junto com a fonte de sinal *(*Wi-Fi), enquanto que os outros pontos encontram-se em um ambiente a parte.

5.2 SITUAÇÕES TESTADAS

A primeira solicitação de sinal pelo cliente foi realizada com o veículo localizado no ponto inicial da trajetória. O resultado pode ser visualizado na Figura 5.2, onde é possível ver que o ESP8266 detecta o pior nível de intensidade *Wi-Fi* entre todos os componentes (maior distância até a fonte do sinal), sem apresentar proximidade com nenhum ponto de parada e devido a isso, não ocorre atualização no mapa.

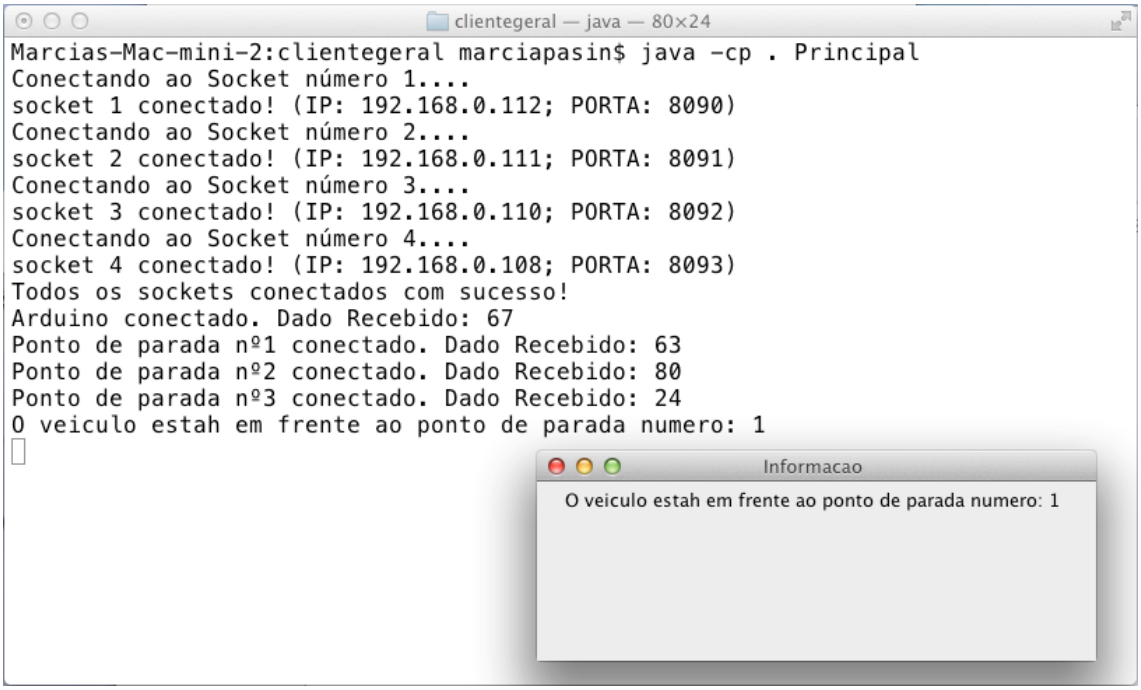
Figura 5.2 – Captura de tela com o sistema em seu estado inicial



Fonte: Autor.

No momento seguinte, o veículo foi posicionado nas proximidades do primeiro ponto de parada, como exemplo desse estado. A leitura dos sinais apresentou uma diferença de 4 DBm entre os componentes (veículo com 67 DBm e o ponto fixo com 63 DBm), ou seja, diferença menor que 8 DBm, indicando que os dois pontos estão muito próximos. O cliente apresenta a informação de que o veículo encontra-se em frente ao ponto de parada em questão (Figura 5.3) e a primeira linha do arquivo tem seu valor modificado de 0 para 1.

Figura 5.3 – Captura de tela com o veículo em frente ao primeiro ponto estático



Fonte: Autor.

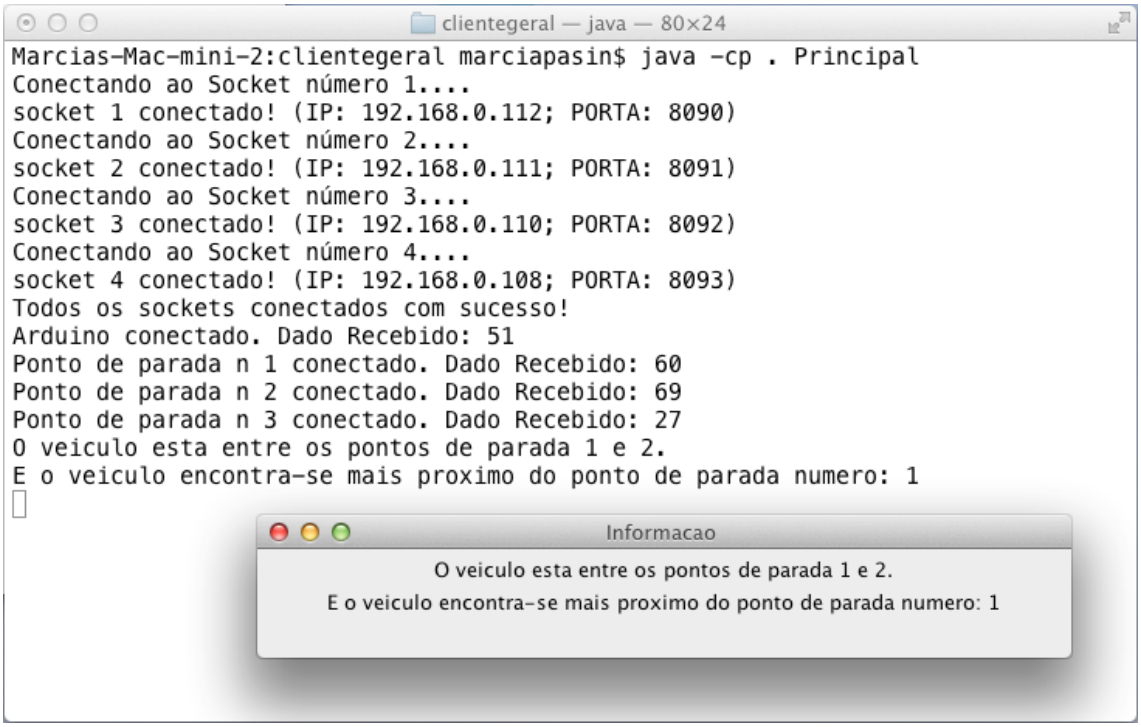
O próximo estado possível é no momento em que o veículo encontra-se entre dois pontos, como pode ser exemplificado na Figura 5.4, com o veículo posicionado entre os pontos estáticos $p1$ e $p2$. A detecção desse estado é feita com base nas seguintes informações:

a) o veículo não se encontra próximo (em frente) a nenhum ponto, já que a diferença de sinal não é menor que 8 DBm para nenhum deles;
b) o arquivo que representa o mapa mostra que o veículo já passou pelo primeiro ponto de parada, mas não passou pelo segundo (primeira linha possui o valor 1, segunda linha possui o valor 0).

É mostrada ainda a informação de que o veículo encontra-se mais próximo do ponto estático $p1$, pois a diferença da intensidade do sinal *Wi-Fi* em relação a esse ponto é a menor em relação aos pontos restantes (9 DBm). Neste estado não há atualização do mapa. O último estado da simulação (Figura 5.5) é quando o veículo já passou por todos os pontos de parada. Neste momento, o mapa possui todas as linhas com o valor 1 e o veículo não se encontra nas proximidades de nenhum ponto. Nota-se que o ESP8266 possui a melhor intensidade do sinal WiFi e também encontra-se afastado do último ponto (diferença de 10 DBm).

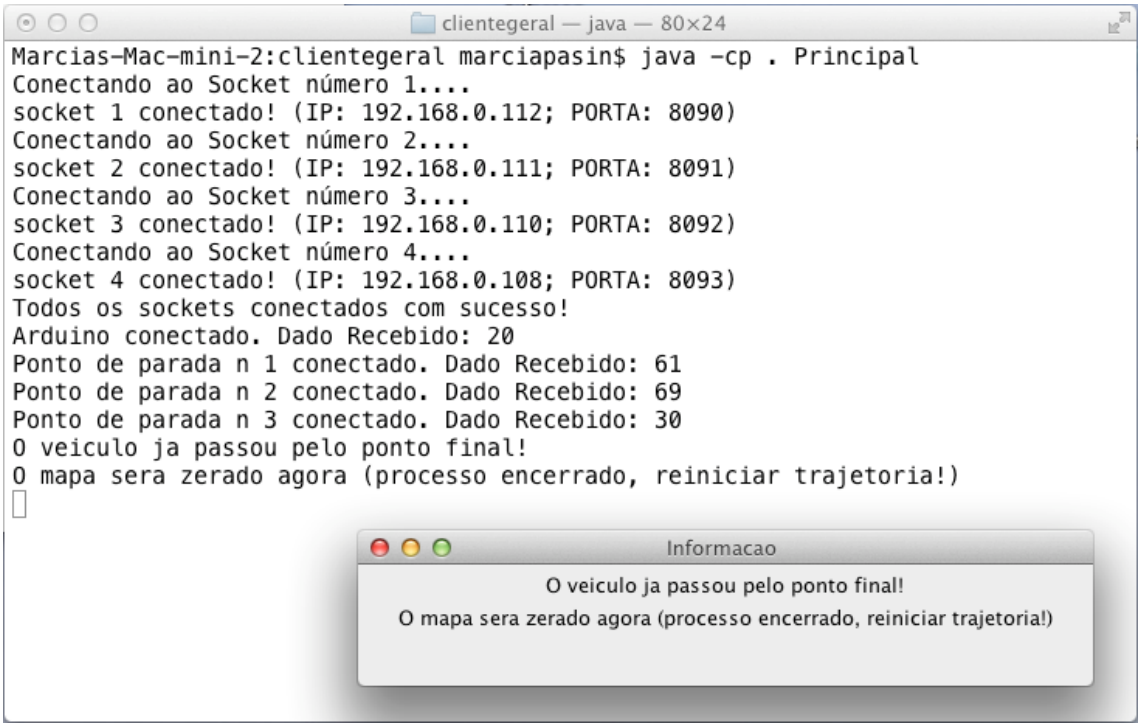
Quando ocorre essa detecção, o mapa é zerado automaticamente para evitar

Figura 5.4 – Captura de tela com o veículo entre os pontos *p1* e *p2*



Fonte: Autor.

Figura 5.5 – Captura de tela após detecção do último estado da simulação



Fonte: Autor.

modificações manuais no arquivo e uma nova simulação pode ser iniciada (essa informação também é apresentada no cliente).

Como é possível ver ao comparar cada execução, existe sempre uma variância significativa na intensidade do sinal em cada componente, como por exemplo, o ponto de parada 3 (três) varia de 24 a 30 DBm durante a simulação. Isso ocorre devido a existência de ruído no ambiente (interferência de outros sinais) que afeta a propagação do sinal, e tecnicamente afeta a maioria das leituras naquele momento, principalmente as mais distantes da fonte de sinal. Obviamente isso ocasiona erros (apresentados na seção seguinte) e deve ser levado em conta como uma futura melhoria do projeto.

5.3 TESTES IMPRECISOS

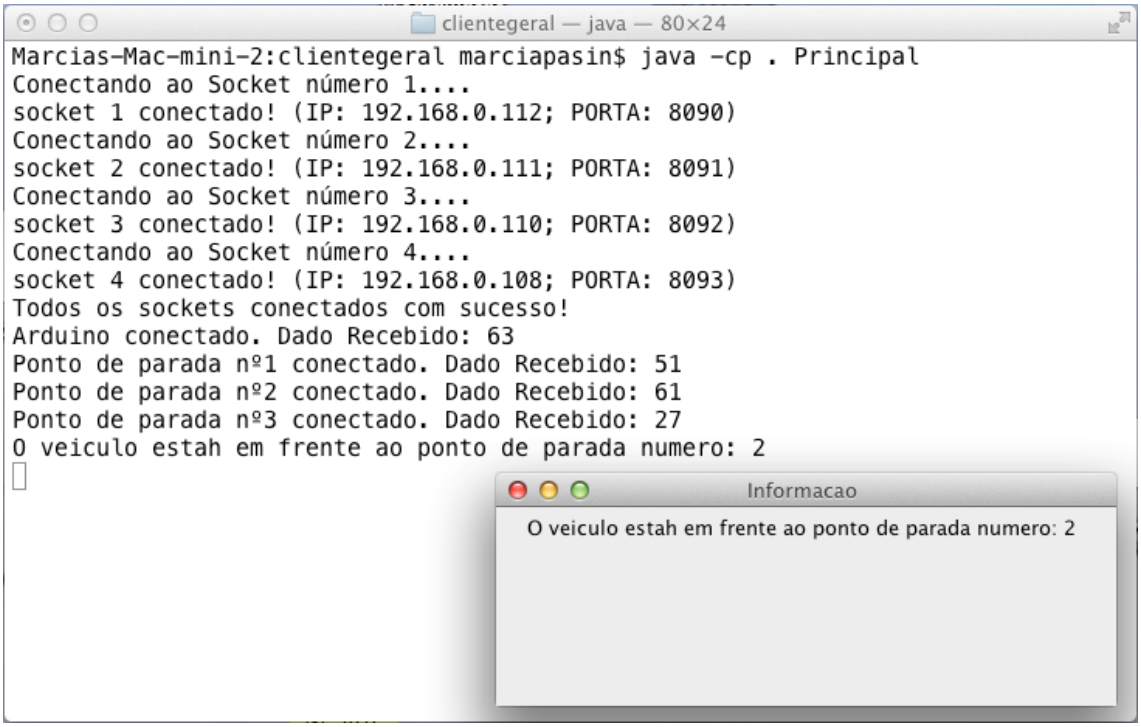
Devido a constante variação nos valores da intensidade do sinal *Wi-Fi* nos pontos estáticos, houve momentos em que a informação apresentada não foi correta, fato esse que já era esperado. Por motivos de ruídos e interferências locais (dados não abordados nesta implementação), ocorrem alterações súbitas e momentâneas na métrica principal em um ou mais pontos, tornando o conjunto de dados inconstante, fazendo com que o programa apresente respostas incoerentes.

Um fator que também influencia no resultado é a captação do sinal pelo módulo ESP8266, que ocorre por uma antena embutida na própria placa. Esse sistema prioriza a captação em apenas um sentido (parte frontal da placa), ou seja, a informação sobre a intensidade do sinal no carro robô é influenciada pela simples rotação do veículo. Testes mostraram que uma rotação de 180º no veículo (mantendo-o na mesma posição), causa uma variância média de 10 DBm na métrica.

Tomando como exemplo o teste apresentado na Figura 5.6, é possível notar que houve uma variância considerável na intensidade do sinal *Wi-Fi* nos pontos *p2* (que anteriormente apresentou o valor 80 e neste teste apresentou o valor 61) e *p3* (que anteriormente apresentou o valor 64 e neste teste apresentou o valor 51). O carro robô encontrava-se em frente ao ponto *p1* e como não sofreu variância significativa na métrica, houve falha na informação final, já que o algoritmo constatou que o veículo encontrava-se em frente ao ponto *p2* (diferença de 2 DBm).

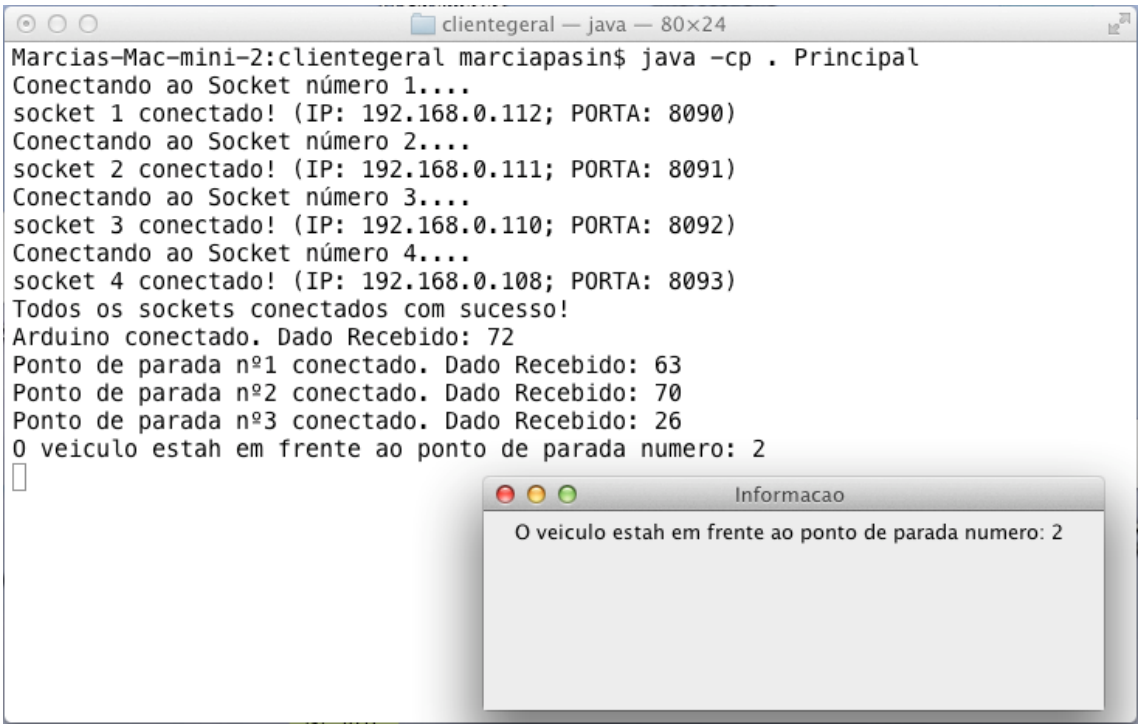
Em outro caso, temos o veículo posicionado entre os pontos *p1* e *p2* e igualmente distante destes pontos. A métrica apresentou um valor mais baixo que o esperado no ponto *p2*, e a placa ESP8266, devido ao sentido do trajeto, estava com sua parte frontal (antena) apontada para o lado oposto à fonte de sinal, causando perda na intensidade do sinal *Wi-Fi* (aumento no valor da métrica). Tais fatos contribuíram para que a informação apresentada não descrevesse corretamente a posição do carro robô no cenário (Figura 5.7).

Figura 5.6 – A informação correta seria o veículo estar em frente ao ponto *p1*



Fonte: Autor.

Figura 5.7 – A informação correta seria o veículo estar entre os pontos *p1* e *p2*



Fonte: Autor.

6 CONCLUSÃO

Este trabalho teve como objetivo apresentar o desenvolvimento em ambiente reduzido, de uma implementação estruturada para ambientes reais, sobre o cenário de um ônibus de transporte público urbano trafegando por seu trajeto de rotina, em comunicação com os pontos de parada, com a finalidade de informar sua localização aproximada aos usuários, fazendo uso de sinal *Wi-Fi*. O veículo (principal ponto de estudo) tem sua estrutura de comunicação baseada na plataforma Arduino, em conjunto com o módulo de comunicação *Wi-Fi* ESP8266 que garantem a conexão com o cliente do serviço e a detecção da métrica principal no veículo (intensidade do sinal).

Durante o desenvolvimento do trabalho, a plataforma Arduino mostrou ter grande potencial para o desenvolvimento de ambientes de teste para estudo e aplicação de técnicas, tendo em vista o baixo custo de aquisição, a possibilidade de trabalhar em conjunto com outros sistemas (como no caso, o ESP8266) e a variedade de materiais sobre programação do módulo, disponíveis na *Web* (em grande parte disponibilizados pelo próprio fabricante).

6.1 LIÇÕES APRENDIDAS

Diversas dificuldades foram encontradas no decorrer do trabalho, devido ao conhecimento limitado do acadêmico com a plataforma do veículo. No início, diversos problemas ocorreram com a programação Arduino no sistema operacional Windows. Não havia, por exemplo, reconhecimento da porta serial de comunicação *Wi-Fi* (Serial1) pela IDE, e após diversas tentativas de configuração sem sucesso, houve a necessidade de migrar para a plataforma Linux, onde obteve-se êxito. A formulação do sistema de comunicação em conjunto com a implementação, apresentou-se como um desafio do trabalho, já que se fez necessário um estudo sobre a plataforma Arduino e sistemas operacionais envolvidos, buscando meios para obter informações a respeito da conexão.

O trabalho com o ambiente de prototipação físico proporciona a experiência de unir teoria e prática e o baixo custo na realização de testes, facilitando a detecção de problemas durante a implementação e melhorias do sistema, em conjunto com o fato de que, ao mesmo tempo, há uma melhor aproximação com o uso do sistema em um ambiente real, ao contrário do que ocorre com simulações realizadas em ambientes virtuais.

Este projeto possibilitou aplicar os conhecimentos adquiridos durante o curso de Bacharelado em Ciência da Computação da UFSM. Destaca-se as disciplinas de Sistemas Distribuídos, Modelagem e Simulação, Paradigmas de Programação e Estrutura de Dados.

6.2 TRABALHOS FUTUROS

Para o prosseguimento do projeto, é possível realizar uma melhoria na precisão da informação referente a intensidade do sinal *Wi-Fi* nos componentes, buscando meios para detectar o nível de ruído em cada local e aplicar filtros. Dessa forma a variância do sinal seria reduzida de forma significativa e a precisão dos resultados finais seria melhor, evitando assim a ocorrência de erros de mapeamento.

Com a informação sobre a intensidade do sinal melhorada, é possível aplicar algoritmos de mapeamento diretos, possibilitando assim o uso do sistema em um ambiente mais complexo e de maior extensão, com diversas fontes de sinal *Wi-Fi* disponíveis. A triangularização é um bom exemplo, pois possibilita uma precisão quase perfeita na localização, buscando dados a partir das conexões entre o veículo e os pontos fixos, e também obtendo informações apresentadas a partir da conexão entre os próprios pontos fixos.

REFERÊNCIAS BIBLIOGRÁFICAS

ARDUINO. **Arduino Platform**. 2009. Disponível em: <https://www.arduino.cc/>. acesso em: 31 out. 2016.

ARDUINO-REFERENCE. **Arduino Reference**. 2015. Disponível em: <https://www.arduino.cc/en/Reference/Comparison>. Acesso em: 21 out. 2016.

ARDUINO.CC. **Arduino MEGA 2560 - Overview.** 2015. Disponível em: <https://www.arduino.cc/en/Main/ArduinoBoardMega2560>. Acesso em: 17 out. 2016.

BARBIERI, W. R. **Proposta e modelagem de comunicação segura para redes veiculares (VANETs)**. 2012. Tese (Doutorado) — Universidade Estadual Paulista Júlio de Mesquita Filho, 2012.

BIEM, A. et al. Ibm infosphere streams for scalable, real-time, intelligent transportation services. In: ACM. **Proceedings of the 2010 ACM SIGMOD International Conference on Management of data**. [S.l.], 2010. p. 1093–1104.

BOUKERCHE, A. et al. Vehicular ad hoc networks: A new challenge for localization-based systems. **Computer communications**, Elsevier, v. 31, n. 12, p. 2838–2849, 2008.

BR-ARDUINO. **Melhor canal para o WiFi da sua casa, o Arduino e o ESP8266 indicam.** 2015. Disponível em: <http://br-arduino.org/2015/07/arduino-esp8266-canais-wifi.html>. Acesso em: 05 out. 2016.

BUSALERT. **Conheça o Bus Alert.** 2016. Disponível em: <http://www.busalert.com.br/page/homel>. Acesso em: 25 set. 2016.

CHEN, Z. D.; KUNG, H.; VLAH, D. Ad hoc relay wireless networks over moving vehicles on highways. In: ACM. **Proceedings of the 2nd ACM international symposium on Mobile ad hoc networking & computing**. [S.l.], 2001. p. 247–250.

CITTAMOBI. **Cittamobi *website*.** 2015. Disponível em: <https://citymapper.com>. Acesso em: 25 set. 2016.

CITYMAPPER. **Citymapper *website*.** 2016. Disponível em: <https://citymapper.com>. Acesso em: 25 set. 2016.

CONRAD, N. **Position Tracking Using WiFi**. 2014. Western Michigan University, 2014.

HARTENSTEIN, H.; LABERTEAUX, L. A tutorial survey on vehicular ad hoc networks. **IEEE Communications magazine**, IEEE, v. 46, 2008.

INSTRUCTABLES. **Arduino Modules - L298N Dual H-Bridge Motor Controller.** 2014. Disponível em: <http://www.instructables.com/id/Arduino-Modules-L298N-Dual-H-Bridge-Motor-Controll/>. Acesso em: 17 out. 2016.

LI, F.; WANG, Y. Routing in vehicular ad hoc networks: A survey. **IEEE Vehicular technology magazine**, IEEE, v. 2, 2007.

LINUX. **The Linux Documentation Project**. 2001. Disponível em: <http://www.tldp.org/LDP/Linux-Filesystem-Hierarchy/html/proc.html>. Acesso em: 25 out. 2016.

MOOVIT. **Moovit *Web app*.** 2016. Disponível em: <https://moovitapp.com>. Acesso em: 25 set. 2016.

OPT. **Lisboa MOVE ME *Web app*.** 2016. Disponível em: <http://lisboa.move-me.mobi/>. Acesso em: 25 set. 2016.

SAVVIDES, A.; HAN, C.-C.; STRIVASTAVA, M. B. Dynamic fine-grained localization in ad-hoc networks of sensors. In: ACM. **Proceedings of the 7th annual international conference on Mobile computing and networking**. [S.l.], 2001. p. 166–179.

SHIELDS-MANUAL. **Arduino Shields Manual**. 2010. Disponível em <http://www.robotshop.com/media/files/PDF/dfrobot-arduino-shields-manual.pdf>. Acesso em: 21 out. 2016.

SUSSMAN, J. **Introduction to transportation systems**. 2000.

TACOM. **Siu Mobile *Website*.** 2016. Disponível em: <http://www.siumobile.com.br/>. Acesso em: 25 set. 2016.

TAN, K.; WONG, K. Low-cost campus bus tracker using wifi access points. In: IEEE. **Consumer Electronics-Taiwan (ICCE-TW), 2016 IEEE International Conference on**. [S.l.], 2016.

TRAFI, T. **Trafi *Website*. World's Most Accurate Public Transport app.** 2016. Disponível em: <https://trafi.com>. Acesso em: 25 set. 2016.

VACCARI, L. S.; FANINI, V. Mobilidade urbana. **Série de cadernos técnicos da agenda parlamentar. Conselho Regional de Engenharia, Arquitetura e Agronomia do Paraná–CREA/PR**, 2010.

APÊNDICE A – CÓDIGO DO ARDUINO

A.1 SKETCH_WIFI.INO

```
#include "ESP8266.h"
#include <NewPing.h>
#include <string.h>
#include "SoftwareSerial.h"

#define SSID "dlinkcars"
#define PASSWORD "aaa134bbb"
#define MAX_DISTANCE 200
#define trigPin A0
#define echoPin A1
#define debug true
#define HOST_PORT   (8090)

ESP8266 wifi(Serial1);

void setup(void)
{
    Serial.begin(9600);
    Serial1.begin(9600);
    setupUltrasonic();
    Serial.println("Iniciando Setup.");
    Serial.print("Versao de Firmware ESP8266: ");
    Serial.println(wifi.getVersion().c_str());
    if (wifi.setOprToStationSoftAP()) {
      Serial.println("Station e AP OK.");
    } else {
       Serial.println("Erro em setar Station e AP - tenta outra vez pois
          pode ser o tera.");
       Serial1.begin(19200);
      if (wifi.setOprToStationSoftAP()) {
         Serial.println("Station e AP OK.");
      } else exit(1);
      }
    if (wifi.joinAP(SSID, PASSWORD)) {
        Serial.println("Conectado com Sucesso.");
        Serial.println("IP: ");
        Serial.println(wifi.getLocalIP().c_str());
        } else {
          Serial.println("Falha na conexao AP.");
      }
    if (wifi.enableMUX()) {
```

```
41          Serial.println("Multiplas conexoes OK.");
42      } else {
43          Serial.println("Erro ao setar multiplas conexoes.");
44      }
45      if (wifi.startTCPServer(HOST_PORT)) {
46          Serial.println("Servidor iniciado com sucesso.");
47      } else {
48          Serial.println("Erro ao iniciar servidor.");
49      }
50      Serial.println("Setup finalizado!");
51  }
52
53  String ComandoAT(String comando, const int timeout)
54  {
55      String resposta = "";
56      String sinal = "";
57      Serial1.println(comando);
58      long int time = millis();
59      char c;
60      char aux = '-';
61      while ( (time + timeout) > millis())
62      {
63          while (Serial1.available())
64          {
65              c = Serial1.read();
66              resposta += c;
67          }
68      }
69      for (int i=0; i<resposta.length() ; i++) {
70          c = resposta.charAt(i);
71          if (c == aux){
72              i++;
73              c = resposta.charAt(i);
74              sinal +=c;
75              i++;
76              c = resposta.charAt(i);
77              sinal +=c;
78              return sinal;
79          }
80      }
81
82      return "Erro no Comando AT";
83  }
84
85  void loop(void) {
86     uint8_t buffer[128] = {0};
87     static uint8_t mux_id = 0;
```

```
88      String sinal = ComandoAT("AT+CWLAP=\"dlincars\",\"c8:3a:35:45:a5:78\",6"
            , 1000);
89      Serial.println("O sinal eh: " +sinal);
90      char si[]= {sinal[0], sinal[1], '\n'};
91      char *request = si;
92      wifi.send(mux_id, (const uint8_t*)si, strlen(si));
93      uint32_t len = wifi.recv(mux_id, buffer, sizeof(buffer), 1000);
94      if (len > 0) {
95         Serial.print(len);
96         Serial.print("Received: [");
97         for(uint32_t i = 0; i < len; i++) {
98           Serial.print((char) buffer[i]);
99         }
100        Serial.print("]\n");
101     }
102     delay (1);
103 }
```

APÊNDICE B – CÓDIGOS DO CLIENTE

B.1 CONNECTION.JAVA

```
1  package clientegeral;
2
3  import java.io.BufferedReader;
4  import java.io.IOException;
5  import java.io.InputStreamReader;
6  import java.net.Socket;
7  import java.util.logging.Level;
8  import java.util.logging.Logger;
9
10 /**
11  *
12  * @author Jeferson Farias da Silva
13  * jfarias@inf.ufsm.br
14  */
15 public class Connection extends Thread{
16     private Socket[] skt;
17     private int socketnum;
18     public int caracter;
19     private int wifipc;
20     private int valor_calibracao = 8;
21
22     public Connection(Socket []skt_){
23         skt = skt_;
24     }
25     @Override
26     public void run(){
27         int i, aux, sok = 0;
28         String chegada, a, b;
29         int[] message = new int[skt.length];
30         BufferedReader in;
31         Jlabel janela = new Jlabel("","");
32
33         try {
34
35                 for(aux=0; aux<skt.length; aux++){
36                     if(aux==0)System.out.print("Arduino conectado. ");
37                     else System.out.print("Ponto de parada numero" + aux +
                           " conectado. ");
38
39                     in = new BufferedReader(new InputStreamReader(skt[aux].
                           getInputStream()));
```

```
40                      chegada = in.readLine();
41                      System.out.println("Dado Recebido: " + chegada);
42                      message[aux] = Integer.parseInt(chegada);
43                  }
44                 if(skt.length>=2){
45                      i=99;
46                      for(aux=1; aux<skt.length; aux++){
47                          if((Math.abs(message[0]-message[aux]))<i){
48                              i=Math.abs(message[0]-message[aux]);
49                              sok=aux;
50                          }
51                      }
52                      if(i<valor_calibracao){
53                          a = "O veiculo esta em frente ao ponto de parada
                                numero: " + String.valueOf(sok);
54                          System.out.println(a);
55                          MapControl.escritor("map.txt", (skt.length-1), sok);
56                          janela = new Jlabel(a,"");
57                          janela.CriaFrame();
58
59                      }else{
60                          aux = MapControl.leitor("map.txt", (skt.length-1));
61
62                          if(aux!= -1){
63                              if(aux == 1){
64                                  a = "O veiculo ainda nao passou pelo
                                        primeiro ponto de parada";
65                                  System.out.println(a);
66                                  janela = new Jlabel(a,"");
67                                  janela.CriaFrame();
68                              }else{
69                                  a = "O veiculo estah entre os pontos de
                                        parada" + String.valueOf((aux-1)) + "e" +
                                         String.valueOf(aux) + ".";
70                                  b = "E o veiculo encontra-se mais proximo do
                                         ponto de parada numero: " + String.
                                        valueOf(sok);
71                                  System.out.println(a);
72                                  System.out.println(b);
73                                  janela = new Jlabel(a,b);
74                                  janela.CriaFrame();
75                              }
76                          }else{
77                              a = "O veiculo ja passou pelo ponto final!";
78                              b = "O mapa sera zerado agora (processo
                                    encerrado, reiniciar trajetoria!)";
79                              System.out.println(a);
```

```
80                              System.out.println(b);
81                              MapControl.resetMap("map.txt", (skt.length-1));
                                    //Arduino nao entra no mapa
82                              janela = new Jlabel(a,b);
83                              janela.CriaFrame();
84                          }
85                      }
86                  } else {
87                      System.out.println("Ha apenas um ponto de parada
                            registrado...");
88                  }
89
90          } catch (IOException ex) {
91              Logger.getLogger(Connection.class.getName()).log(Level.SEVERE,
                    null, ex);
92          }
93      }
94
95  }
```

B.2 JLABEL.JAVA

```
1   package clientegeral;
2   import javax.swing.*;
3
4   /**
5    *
6    * @author Jeferson Farias da Silva
7    * jfarias@inf.ufsm.br
8    */
9   public class Jlabel {
10
11      public JFrame f;
12      public JLabel texto;
13      public JLabel texto2;
14      public JPanel p;
15
16      public Jlabel(String a, String b) {
17          f = new JFrame("Informacao ");
18          texto = new JLabel(a);
19          texto2 = new JLabel(b);
20          p = new JPanel();
21      }
22
23      public void CriaFrame() {
```

```
24          f.setSize(400, 150);
25          f.setLocationRelativeTo( null );
26          f.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);
27          p.add(texto);
28          p.add(texto2);
29          f.add(p);
30          f.setVisible(true);
31      }
32
33  }
```

B.3 MAIN.JAVA

```
1   package clientegeral;
2
3   import java.io.IOException;
4   import java.net.ConnectException;
5   import java.net.Socket;
6
7   /**
8    *
9    * @author Jeferson Farias da Silva
10   * jfarias@inf.ufsm.br
11   */
12  public class Main {
13      private static final String IP[] = {"192.168.0.112","192.168.0.110","
            192.168.0.108","192.168.0.111"};
14      private static final int[] PORT = {8090, 8091, 8092, 8093};
15      /**
16       * @param args the command line arguments
17       */
18      public static void main(String[] args) {
19          int socketnum = 1;
20          Connection c1 = null;
21          Socket[] skt = new Socket[IP.length];
22          int diff;
23          int aux = 0;
24          try {
25                  for(aux=0; aux<IP.length; aux++){
26                      System.out.println("Conectando ao Socket numero " + (
                            aux+1) + "....");
27                      skt[aux] = new Socket(IP[aux], PORT[aux]);
28                      System.out.println("socket " + (aux+1) + " conectado! (
                            IP: " + IP[aux] +"; PORTA: "+ PORT[aux] + ")");
29
```

```
30                  }
31              System.out.println("Todos os sockets conectados com sucesso!");
32              c1 = new Connection(skt);
33              c1.start();
34              Thread.sleep(100);
35
36          } catch (ConnectException e)  {
37              System.out.println("Nao foi possivel conectar ao Socket " + (
                    aux+1));
38          } catch (IOException e) {
39              System.out.println("NoRouteToHost IOException " + (aux+1));
40          } catch (InterruptedException e) {
41              System.out.println("NoRouteToHost InterruptedException " + (aux
                    +1));
42          }
43      }
44
45
46  }
```

B.4 MAPCONTROL.JAVA

```
1   package clientegeral;
2
3   import java.io.BufferedReader;
4   import java.io.BufferedWriter;
5   import java.io.FileReader;
6   import java.io.FileWriter;
7   import java.io.File;
8
9   /**
10   *
11   * @author Jeferson Farias da Silva
12   * jfarias@inf.ufsm.br
13   */
14  public class MapControl {
15   public static int leitor(String path, int nlinhas) throws IOException {
16          BufferedReader buffRead = new BufferedReader(new FileReader(path));
17          int aux = -1;
18          int aux1 = -1;
19          String comp = "0";
20          String linha = "";
21          linha = buffRead.readLine();
22          for(aux=1; aux<=nlinhas; aux++){
23              if (linha.equals(comp)){
```

```
24                  aux1=aux;
25                  break;
26              } else{
27              linha = buffRead.readLine();
28              }
29          }
30          buffRead.close();
31          return aux1;
32      }
33
34      public static void escritor(String path, int nlinhas, int linhaAlterar)
            throws IOException {
35          String[] skt = new String[nlinhas];
36          BufferedReader buffRead = new BufferedReader(new FileReader(path));
37          String linha = "";
38          int aux = 1;
39          while ((aux) <= nlinhas) {
40              linha = buffRead.readLine();
41              skt[aux-1] = linha;
42              aux++;
43          }
44          skt[linhaAlterar-1]="1";
45          buffRead.close();
46          BufferedWriter buffWrite = new BufferedWriter(new FileWriter(path,
                false));
47          aux = 1;
48          while ((aux) <= nlinhas) {
49              buffWrite.write(skt[aux-1]);
50              buffWrite.newLine();
51              aux++;
52          }
53          buffWrite.close();
54      }
55
56          public static void resetMap(String path, int nlinhas) throws
                IOException {
57          BufferedWriter buffWrite = new BufferedWriter(new FileWriter(path,
                false));
58          int aux = 1;
59          while ((aux) <= nlinhas) {
60              buffWrite.write("0");
61              buffWrite.newLine();
62              aux++;
63          }
64          buffWrite.close();
65      }
66  }
```

APÊNDICE C – CÓDIGOS DOS SERVIDORES LINUX

C.1 SINAL_LINUX.JAVA

```
1  package sinal_linux;
2
3  import java.net.*;
4  import java.io.*;
5  import java.net.Socket;
6
7  /**
8   *
9   * @author Jeferson Farias da Silva
10  * jfarias@inf.ufsm.br
11  */
12 public class Sinal_Linux implements Runnable {
13     ServerSocket ss;
14     int i = 3;
15     int aux = 0;
16     TrataCliente mensagem;
17     int clientes[] = new int[10];
18
19     public Sinal_Linux(int porta) throws Exception {
20         ss = new ServerSocket(porta);
21         new Thread(this).start();
22         System.out.println("Servidor aberto na porta:" + porta);
23     }
24
25     public void run() {
26             String addr;
27             Socket clie;
28         try {
29             while(true) {
30                 clie=ss.accept();
31                 mensagem = new TrataCliente(clie);//.start();
32                 mensagem.start();
33                 System.out.println("Mais um cliente atendido!");
34                 aux++;
35             }
36         } catch (Exception e) {
37             e.printStackTrace();
38             System.exit(1);
39         }
40     }
41     public static void main(String[] args) {
```

```
42          try {
43              new Sinal_Linux(8092);
44          } catch (Exception e) {
45              e.printStackTrace();
46              System.exit(1);
47          }
48      }
49  }
50  class TrataCliente extends Thread {
51      private Socket client;
52      public String dado_in;
53      public Wifintensity info = null;
54      public TrataCliente(Socket s) {
55          client = s;
56      }
57      public void run() {
58          try {
59              PrintWriter out = new PrintWriter(client.getOutputStream(),
                    true);
60              info = new Wifintensity();
61              out.println(info.d3);
62              client.close();
63          } catch (Exception e) {
64              e.printStackTrace();
65              System.exit(1);
66          }
67      }
68  }
```

C.2 WIFINTENSITY.JAVA

```
1   package sinal_linux;
2
3   import java.io.BufferedReader;
4   import java.io.FileNotFoundException;
5   import java.io.FileReader;
6   import java.io.IOException;
7
8   /**
9    *
10   * @author Jeferson Farias da Silva
11   * jfarias@inf.ufsm.br
12   */
13  public class Wifintensity {
14      FileReader fileR;
```

```
15      BufferedReader buff;
16      int caracter = 0;
17      char d1, d2;
18      String d3;
19
20      public Wifintensity() {
21          try {
22              fileR = new FileReader("/proc/net/wireless");
23              buff = new BufferedReader(fileR);
24              buff.readLine();
25              buff.readLine();
26
27              while ((caracter = buff.read()) != 45) {
28              }
29              caracter = buff.read();
30              d1 = (char)caracter;
31              caracter = buff.read();
32              d2 = (char)caracter;
33              StringBuilder aux = new StringBuilder();
34              aux.append(d1);
35              aux.append(d2);
36              d3 = aux.toString();
37              caracter = Integer.parseInt(d3);
38              System.out.println("Sinal wifi atual no PC: -" + (caracter));
39              buff.close();
40
41          } catch (FileNotFoundException ex) {
42              System.out.println("Falha no arquivo");
43          } catch (IOException er) {
44              System.out.println("Falha no arquivo");
45
46          }
47      }
48
49  }
```

APÊNDICE D – CÓDIGO DO SERVIDOR MAC

D.1 SINAL_MACOS.JAVA

```
1  import java.net.*;
2  import java.io.*;
3  import java.net.Socket;
4
5  /**
6   *
7   * @author Jeferson Farias da Silva
8   * jfarias@inf.ufsm.br
9   */
10 public class Sinal_MacOS implements Runnable {
11     ServerSocket ss;
12     int i = 3;
13     int aux = 0;
14     TrataCliente mensagem;
15     int clientes[] = new int[10];
16
17     public Sinal_MacOS(int porta) throws Exception {
18         ss = new ServerSocket(porta);
19         new Thread(this).start();
20         System.out.println("Servidor aberto na porta:" + porta);
21     }
22
23     public void run() {
24             String addr;
25             Socket clie;
26
27         try {
28             while(true) {
29                 clie=ss.accept();
30                 mensagem = new TrataCliente(clie);
31                 mensagem.start();
32                 System.out.println("Mais um cliente atendido!");
33             }
34         } catch (Exception e) {
35             e.printStackTrace();
36             System.exit(1);
37         }
38     }
39
40     public static void main(String[] args) {
41         try {
```

```
42              new Sinal_MacOS(8093);
43          } catch (Exception e) {
44              e.printStackTrace();
45              System.exit(1);
46          }
47      }
48  }
49
50  class TrataCliente extends Thread {
51      private Socket client;
52      public String dado_in;
53      boolean success = false;
54      String result;
55      Process p;
56      BufferedReader input;
57      StringBuffer cmdOut = new StringBuffer();
58      String lineOut = null;
59      int numberOfOutline = 0;
60
61      public TrataCliente(Socket s) {
62          client = s;
63      }
64
65      public void run() {
66          try {
67              PrintWriter out = new PrintWriter(client.getOutputStream(),
                    true);
68              String Start ="/System/Library/PrivateFrameworks/Apple80211.
                    framework/Versions/Current/Resources/airport -s";
69              p = Runtime.getRuntime().exec(Start);
70              input = new BufferedReader(new InputStreamReader(p.
                    getInputStream()));
71              while ((lineOut = input.readLine()) != null) {
72                  if (numberOfOutline > 0) {
73                      cmdOut.append("\n");
74                  }
75                  cmdOut.append(lineOut);
76                  numberOfOutline++;
77              }
78              result = cmdOut.toString();
79              System.out.println("ola.....");
80              String[] split = result.split("\n");
81              for(int i = 0; i < split.length; i++){
82                  boolean retval = split[i].contains("dlinkcars");
83                  if(retval == true)
84                      result = split[i];
85              }
```

```
86              result = result.trim();
87              result = result.substring(29, 32);
88              System.out.println("em string "+ result);
89              int valor = Integer.parseInt(result.trim());
90              System.out.println("em inteiro "+ valor);
91              success = true;
92              input.close();
93              out.println(valor);
94              client.close();
95          } catch (Exception e) {
96              e.printStackTrace();
97              System.exit(1);
98          }
99      }
100 }
```