ALISSON HISAO KOBORI

WELLINGTON BIING JUNG LEE

PROJETO E IMPLEMENTAÇÃO DE UMA APLICAÇÃO PARA ANÚNCIO DE PARADAS EM LINHAS DE ÔNIBUS

Monografia apresentada à Escola Politécnica da Universidade de São Paulo para a conclusão de curso.

Curso de Graduação:

Engenharia Mecatrônica

Orientador:

Prof. Dr. Fabrício Junqueira

São Paulo

2016

ALISSON HISAO KOBORI

WELLINGTON BIING JUNG LEE

PROJETO E IMPLEMENTAÇÃO DE UMA APLICAÇÃO PARA ANÚNCIO DE PARADAS EM LINHAS DE ÔNIBUS

Monografia apresentada à Escola Politécnica da Universidade de São Paulo para a conclusão de curso.

Curso de Graduação:

Engenharia Mecatrônica

Orientador:

Prof. Dr. Fabrício Junqueira

São Paulo

2016

Catalogação-na-publicação

Kobori, Alisson Hisao; Lee, Wellington Biing Jung
Projeto e implementação de uma aplicação para anúncio de paradas em linhas de ônibus / A.H. Kobori; W.B.J. Lee – São Paulo, 2016
46 p.

Trabalho de Formatura - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos.

1.Sistema RTPI 2.GPS 3.Internet of Things I.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos II.t.

AGRADECIMENTOS

Ao nosso orientador Prof. Dr. Fabrício Junqueira pela sua constante orientação, paciência durante o período da realização do trabalho e pela ajuda nos momentos difíceis.

Aos professores da Escola Politécnica da USP, por terem nos passado seu conhecimento e nos ensinado a enfrentar desafios.

A todos aqueles que contribuíram direta ou indiretamente na produção deste trabalho.

RESUMO

Devido à falta de informações dentro dos ônibus sobre as linhas percorridas, muitos usuários têm dificuldade em saber a localização exata do veículo e ficam em dúvida em relação à qual parada desembarcar, criando uma dificuldade no planejamento de sua viagem. O objetivo deste trabalho é projetar e implementar um sistema que informe aos usuários de um ônibus as próximas paradas e o tempo de viagem estimado até elas. Para tanto, o sistema é controlado por um *hardware* baseado em microcontrolador, capaz de coletar dados via GPS, cadastrar informações da linha de ônibus e realizar anúncios através de um monitor de LCD e um alto-falante. Além disso, desenvolveu-se um *software* capaz de processar os dados recebidos, realizar cálculos para informar a localização das próximas paradas e estimar o tempo de viagem. A configuração, início e finalização do sistema são realizados por um operador que é responsável por inserir os dados da linha percorrida pelo ônibus. O sistema executa a maior parte das ações e possui a função de passar as informações ao usuário de maneira visual e sonora. A arquitetura é baseada em *Internet of Things* na qual a interconexão *hardware-software* é feita via Internet, ou seja, os dados e algoritmos desenvolvidos no *software* são transferidos ao microcontrolador pela rede.

Sumário

1. Introdução 1
1.1. Objetivos do trabalho 3
1.2. Metodologia Utilizada 4
2. Revisão Bibliográfica 5
2.1. A plataforma Arduino 5
2.2. A plataforma Raspberry Pi 7
2.3. Estrutura do hardware 8
2.3.1. Sistema de anúncio de paradas de maneira sonora e visual 8
2.3.2. Coleta de dados por GPS 8
2.3.3. Armazenamento do itinerário e das mensagens aos usuários 9
2.4. Projeto do software 10
2.4.1. Cálculo da posição e estimativa de tempo até a próxima parada 10
2.4.2. Gerenciamento de mensagens no monitor e no alto-falante 10
2.5. Escolha do hardware: Arduino x Raspberry Pi 11
2.6. Algoritmo de GPS: cálculo da distância e do tempo de viagem 11
3. Descrição do Sistema Proposto 16
3.1. Diagrama de Casos de Uso 16
3.2. Diagrama de Atividades 18
3.2.1. Gerenciar operação 18
3.2.2. Selecionar itinerário 19
3.3. Diagrama de Classes 20
3.3.1. Mensagem 21
3.3.2. Itinerário 21
3.3.3. Modulo_GPS 22
3.3.4. Monitor_LCD 22
3.3.5. Alto_falante 22
3.3.6. Microcontrolador 23
3.4. Diagrama de Sequência 23
4. Teste do Sistema de Aquisição de Dados via GPS 26

4.1. Instalação do Módulo GPS no Arduino 26
4.2. Teste do Sistema de Aquisição via GPS 27
5. Implementação do Sistema 29
5.1. Aspectos Gerais 29
5.2. Configuração e Implementação da plataforma 30
5.2.1. Montagem do Sistema 30
5.2.2. Implementação da Interface Gráfica 32
5.2.3. Implementação do Algoritmo de Cálculo das Próximas Paradas e Tempo Estimado 34
6. Conclusões 37
7. Referências Bibliográficas 38

1. Introdução

O ônibus é uma modalidade muito utilizada que possibilita o transporte de uma quantidade elevada de passageiros em um único veículo. Com a política de criação de faixas de ônibus adotada pela Prefeitura de São Paulo no ano de 2013, o número de passageiros que utilizam esse meio de transporte vem aumentando (DO VALLE, 2014). De acordo com os dados da Prefeitura de São Paulo, em março de 2015 foi computada uma média 9,6 milhões de passageiros transportados diariamente por este meio de transporte. A partir desses dados, pode-se afirmar que o sistema de transporte público por ônibus é essencial para a locomoção de pessoas em São Paulo. Entretanto, é frequente os passageiros não saberem sua localização devido à falta de informações da linha de ônibus no decorrer do trajeto. Com isso, necessitam perguntar ao cobrador a parada de destino ou acabam se perdendo no meio do caminho.

Esse problema pode ser minimizado por meio do desenvolvimento de um sistema de informação em tempo real para passageiros (*RTPI system*) no interior dos ônibus, podendo alertar os usuários, de maneira visual e sonora, as próximas paradas bem como o tempo estimado até elas.

Ao providenciar um amplo espectro de informações, os sistemas RTPI já mostraram causar efeitos positivos nos passageiros, tais como a redução no tempo de espera, fatores psicológicos positivos (como a redução da incerteza, facilidade de planejamento e sentimento de segurança), disposição para pagar pelo transporte público e maior satisfação dos usuários. Além disso, oferecem melhorias significativas para o sistema de transporte em geral, uma vez que influenciam as pessoas para o uso de meios de transporte públicos – o que reduz o número de carros nas ruas, diminuindo, assim, os congestionamentos (DZIEKAN e KOTTENHOFF, 2006; POLITIS, PAPAIOANNOU, BASBAS e DIMITRIADIS, 2010).

Na cidade de Kyoto (Japão) os ônibus da cidade possuem um sistema RTPI deste tipo. Em monitores localizados em lugares estratégicos do veículo (Figura 1) sinaliza-se para o passageiro informações como: os próximos 4 pontos de parada, se o ônibus irá parar no próximo ponto ou não e o preço das passagens. Além disso, por meio sonoro, o motorista é capaz de avisar sobre o próximo ponto de parada bem como possíveis atrasos devido às interferências no trajeto.

Figura 1 – Sistema de avisos dos ônibus de Kyoto.

No Brasil, há um projeto deste tipo na cidade de São Paulo. A prefeitura de São Paulo já está testando, por meio de um projeto da SPTrans, a implementação de novas tecnologias nos ônibus da cidade. Entre essas novas tecnologias estão inclusas: alto-falantes internos e externos, que avisam os passageiros sobre as próximas paradas; um painel eletrônico, que informa o itinerário para facilitar o desembarque do passageiro e sistema de GPS, que permite saber a localização instantânea do veículo e até informar os usuários em painéis nos pontos de ônibus (Figura 2) (SECRETARIA EXECUTIVA DE COMUNICAÇÃO, 2013).

Figura 2 – Painel eletrônico informando o itinerário aos passageiros. (SECRETARIA EXECUTIVA DE COMUNICAÇÃO, 2013).

1.1. Objetivos do trabalho

Neste sentido, o objetivo deste trabalho é projetar e implementar uma solução de *hardware* e *software* para ser utilizado no interior de um ônibus, visando alertar os usuários sobre as próximas paradas (nome da parada e estimativa de tempo de chegada). Para fins de validação, pretende-se cadastrar (nome e coordenadas) as paradas de ônibus percorridas por uma linha circular da USP. Caso não seja possível testar o projeto em um dos circulares, o intuito é realizar o mesmo trajeto de carro, registrando o percurso e as informações fornecidas pelo aplicativo/sistema.

Para alcançar o objetivo, as metas que precisam ser atingidas ao longo do projeto foram:

- Disponibilizar um sistema que informe aos usuários de um ônibus as próximas paradas e o tempo de viagem estimado.

- Desenvolver uma plataforma controlado pelo *hardware* Raspberry Pi, capaz de coletar dados via GPS, cadastrar informações da linha de ônibus e realizar anúncios por meio de um monitor LCD e um alto-falante.

- Desenvolver um *software* que será implementado na plataforma, capaz de processar os dados recebidos via GPS, realizar cálculos para informar a localização das próximas paradas e estimar o tempo de viagem.

1.2. Metodologia Utilizada

O projeto visa o desenvolvimento de um sistema para a execução de tarefas de modo inteligente, e para tornar isso possível é necessário auxilio de *hardware* e *software* para gerenciar as funcionalidades presentes. Para obter um embasamento teórico, foi realizada uma revisão bibliográfica de soluções existentes, possíveis *hardwares* a serem utilizados, as diferentes aplicações incluídas e algoritmos de cálculo a partir de dados obtidos via GPS.

Antes de iniciar a implementação do projeto de fato, foi preciso verificar a diversas funcionalidades envolvidas. Para ajudar nesse processo de documentação, foi utilizado diagramas UML. Com isso, a implementação do *software* é facilitada e feita de maneira ordenada.

Com base nos diagramas UML projetados, o sistema será implementado e testado na próxima etapa do projeto.

2. Revisão Bibliográfica

O projeto consiste em uma estrutura de *hardware* e *software*. Para o desenvolvimento, foram considerados duas possíveis soluções de *hardware*: o Arduino e o Raspberry Pi. Cada um possui prós e contras que precisam ser levados em consideração no desenvolvimento do projeto. Além disso, é necessário também fazer a aquisição dos dados por GPS e realizar cálculos para obter a localização e o tempo de viagem. Sendo assim, foi feito uma revisão bibliográfica e estudo dos tópicos citados, de maneira a proporcionar o melhor resultado ao projeto e atingir as metas estabelecidas.

2.1. A plataforma Arduino

De acordo com BADAMASI (2014), o Arduino é uma plataforma de prototipagem eletrônica, baseado em *hardware* e *software* de fácil utilização. O modelo mais simples é o Arduino Uno R3 e seu *hardware* é composto pelos seguintes componentes:

- Entrada USB: usado para fazer o *upload* do programa ao microcontrolador e alimentar a placa.
- Fonte de alimentação externa: usado para alimentar a placa e cuja tensão pode ser regulada de 9 a 12 V.
- Botão de *reset*: reseta o Arduino para executar outro comando caso tenha sido feito o *upload*.
- Microcontrolador: recebe e envia as informações de comando ao circuito utilizado.
- Pinos analógicos: pinos de entrada analógica, de A0 a A5.
- Pinos I/O digitais: pinos de entrada e saída digital, de 2 a 13.

O *software* é composto pelo IDE (*Integrated Development Enviroment*) e é dividido em 3 partes principais:

- Área de comando: composto por um menu de itens como *File, Edit, Sketch, Tools, Help* e ícones de verificação e *upload* do programa.
- Área de texto: local para escrever o programa.
- Janela de mensagem: mostra mensagens do IDE relacionadas à verificação do código.

O Arduino Uno R3 possui uma capacidade analógica diversificada, possibilitando a utilização em conjunto com praticamente todos os tipos de componentes, sensores e atuadores. A linguagem de programação utilizada é baseada em C e é de fácil implementação por não estar vinculado a nenhum sistema operacional (ORSINI, 2015).

A placa básica é composta por um controlador Atmel AVR de 8 bits (algumas versões chegam a 32 bits), conexões digitais e analógicas e entrada USB para ligação simples e direta a computadores. A plataforma suporta e transmite uma corrente elétrica de até 40 mA. A placa possui memória RAM de 2 KB e consome 175 mW (SOUSA, 2015).

Devido à existência de diversos modelos de Arduino, foi feita uma análise mais detalhada dessas placas.

Tabela 1 – Comparação entre os diferentes modelos de Arduino (FILIPEFLOP, 2014)

	Arduino Uno	Arduino Mega2560	Arduino Leonardo	Arduino Due	Arduino ADK	Arduino Nano	Arduino Pro Mini	Arduino Esplora
								
Microcontrolador	ATmega328	ATmega2560	ATmega32u4	AT91SAM3X8E	ATmega2560	ATmega168 (versão 2.x) ou ATmega328 (versão 3.x)	ATmega168	ATmega32u4
Portas digitais	14	54	20	54	54	14	14	-
Portas PWM	6	15	7	12	15	6	6	-
Portas analógicas	6	16	12	12	16	8	8	-
Memória	32 K (0,5 K usado pelo bootloader)	256 K (8 K usados pelo bootloader)	32 K (4 K usados pelo bootloader)	512 K disponível para aplicações	256 K (8 K usados pelo bootloader)	16 K (ATmega168) ou 32K (ATmega328), 2 K usados pelo bootloader	16 K (2k usados pelo bootloader)	32 K (4 K usados pelo bootloader)
Clock	16 Mhz	16 Mhz	16 Mhz	84 Mhz	16 Mhz	16 Mhz	8 Mhz (modelo 3.3v) ou 16 Mhz (modelo 5v)	16 Mhz
Conexão	USB	USB	Micro USB	Micro USB	USB	USB Mini-B	Serial / Módulo USB externo	Micro USB
Conector para alimentação externa	Sim	Sim	Sim	Sim	Sim	Não	Não	Não
Tensão de operação	5v	5v	5v	3.3v	5v	5v	3.3v ou 5v, dependendo do modelo	5v
Corrente máxima portas E/S	40 mA	40 mA	40 mA	130 mA	40 mA	40 mA	40 mA	-
Alimentação	7 - 12 Vdc	7 - 12 Vdc	7 - 12 Vdc	7 - 12 Vdc	7 - 12 Vdc	7 - 12 Vdc	3.35 - 12 V (modelo 3.3v), ou 5 - 12 V (modelo 5v)	5v

Pela análise da Tabela 1, percebe-se que a principal diferença entre os modelos de Arduino é o número de portas digitais, PWM e analógicas. A memória varia de 16 a 512 kbytes dependendo do microcontrolador utilizado. Apesar de o Arduino Due possuir a placa com a maior capacidade de processamento, suas portas operam à 3,3 V, o que compromete a utilização de alguns *shields* disponíveis no mercado que trabalham a 5 V. Com exceção ao Arduino Pro Mini, todas as outras placas possuem a mesma capacidade de processamento de 16 MHz (FILIPEFLOP, 2014). Sendo assim, a escolha da placa utilizada dependerá basicamente do número de portas e da memória necessárias.

2.2. A plataforma Raspberry Pi

Segundo RICHARDSON e WALLACE (2013), o Raspberry Pi é um *hardware* de desenvolvimento parecido com os componentes internos de um celular, com diversos conectores acessíveis a várias portas e funções. Seus principais componentes são:

- Processador: baseado em sistema de um chip de 700 MHz e 32 bits, construído sobre a arquitetura ARM11.
- *Slot* para cartão de memória: não há disco rígido, todo o conteúdo é armazenado em um cartão de memória SD.
- Porta USB: a placa original suportava 100 mA de corrente, mas nas versões mais recentes, é possível suportar as especificações completas das portas USB 2.0.
- Porta Ethernet: dependendo do modelo pode possuir padrão RJ45 ou adaptador de rede USB.
- Conector HDMI: oferece saída de áudio e vídeo digital.
- LEDs de *status*: são 5 LEDs que indicam diferentes situações na placa.
- Saída de áudio analógico: conector de áudio analógico padrão, mas que possui qualidade inferior ao HDMI.
- Saída de vídeo composto: conector padrão do tipo RCA e com resolução extremamente baixa quando comparada com o HDMI.
- Entrada de energia: alimentado por um conector micro USB.

O Raspberry Pi não é tão flexível na leitura de sensores analógicos, muitas vezes, necessitando da assistência de um *hardware* extra. O seu funcionamento pode ser vinculado ao sistema operacional Linux e com a programação feita em Python. Recentemente, surgiu a possibilidade de instalar o Windows 10 IoT, uma versão de Windows 10 otimizado para plataformas menores como o Raspberry Pi. Essa ferramenta utiliza a Plataforma Universal Windows (UWP), facilitando o desenvolvimento de aplicações que possibilitam integrar interface com o usuário, pesquisa, armazenamento online e serviços baseado em nuvem (MICROSOFT, 2016). Comparando com o Arduino, a capacidade de processamento de dados é 40 vezes mais rápido, levando em consideração a velocidade do *Clock*. Além disso, o Raspberry Pi possui uma memória RAM 128000 vezes maior. Outro benefício é a capacidade de integrar diversas funções como internet, vídeo e processamento de áudio em uma única placa (ORSINI, 2015).

2.3. Estrutura do *hardware*

Para a estrutura do *hardware,* alguns requisitos devem ser atendidos. O anúncio das próximas paradas será feito de forma sonora através de um alto-falante e, de forma visual, através de um monitor. A aquisição das coordenadas é feita por um receptor GPS. O itinerário com o percurso do ônibus precisa ser armazenado para que os dados sejam processados durante a implementação do *software.* Devido à necessidade de obter resultados instantâneos, a plataforma precisa ser capaz de processar uma grande quantidade de dados.

2.3.1. Sistema de anúncio de paradas de maneira sonora e visual

Segundo o LABORATÓRIO DE GARAGEM (2013), existem diversas maneiras de conectar um alto-falante ao Arduino e elas dependem do formato do áudio utilizado. Para um áudio em formato MP3 é possível utilizar um MP3 *Player Shield* da Sparkfun que já possui um espaço para inserir um cartão SD e uma saída de áudio para conectar uma caixa de som.

Para gerar um anúncio de maneira visual, é preciso conectar o *hardware* utilizado com um monitor LCD. Os dois tipos de saídas mais comuns e que estão presentes em televisões, monitores e *notebooks* são o VGA e o HDMI. O Arduino não possui nenhuma dessas saídas, por isso, para obter uma conexão VGA, por exemplo, é necessário conectar módulos que ofereçam suporte à placa. Um desses módulos é o MicroVGA que auxiliado por bibliotecas e códigos, conecta-se ao Arduino através de portas digitais, proporcionando uma saída VGA e possibilitando mostrar interfaces com cores e textos em um monitor com esse tipo de entrada (MICROVGA, 2012).

O Raspberry Pi já possui saídas de áudio analógico padrão de 3,5 mm destinado a conduzir cargas de alta impedância. Mesmo assim, conectar com fones de ouvido ou alto-falantes sem alimentação não proporciona sons de qualidade. A melhor opção seria usar saída HDMI que além de fornecer uma qualidade sonora melhor, também possibilita a conexão com um monitor (RICHARDSON & WALLACE, 2013) que será utilizado no anúncio de paradas visualmente.

2.3.2. Coleta de dados por GPS

Atualmente, a utilização do GPS para rastreamento de veículos já está bastante difundida. Essa ferramenta pode proporcionar diversas aplicações como geolocalização, estimativa do tempo de percurso e análise do trânsito nos trechos das estradas. Com essa tecnologia é possível desenvolver aplicações para auxiliar na viagem das pessoas por meios de transporte e ajudar a monitorar

o trânsito nos grandes centros urbanos (MEZA, LIZÁRRAGA e LA FUENTE, 2013).

Para que o projeto atenda as metas especificadas, é necessário utilizar um recurso de coleta de dados por GPS e, a partir disso, calcular os parâmetros que indicam se o ônibus chegou na próxima parada e estimar o tempo de viagem até ela.

Existem módulos de GPS que podem ser encaixados no Arduino e no Raspberry Pi para promover a aquisição de dados via GPS. Um exemplo é o *shield* de GPS para o Arduino da Sparkfun, que possui diversas características que tornam fácil o seu uso. Esse *shield* possui 2 soquetes de 8 pinos e 2 de 6 pinos que precisam ser inicialmente soldadas à placa, e um módulo de GPS EM-506 que precisa ser encaixado à placa. Após isso, o *shield* já pode ser encaixado diretamente ao Arduino. Possui um botão liga/desliga que controla a energia fornecida ao módulo e um botão de *reset* (CAVIS, 2015).

Para o Raspberry Pi, existe um módulo de GPS da Adafruit que possui uma função semelhante ao *shield* visto anteriormente. Nesse caso, é necessário adquirir um cabo conversor de TTL para USB, possibilitando o encaixe no Raspberry Pi (TOWNSEND, 2013).

2.3.3. Armazenamento do itinerário e das mensagens aos usuários

O Raspberry Pi já possui um *slot* para armazenamento de dados em um cartão SD e por não possuir um disco rígido, todas as informações serão armazenadas nesse cartão. Outra maneira seria conectar um *pen drive* ou um HD externo contendo as informações necessárias ao projeto (RICHARDSON & WALLACE, 2013). Sendo assim, não é preciso a utilização de módulos ou *shields* no armazenamento de dados, tornando o Raspberry Pi bem vantajoso.

O Arduino já possui uma memória EEPROM interna e seu tamanho varia conforme o tipo do microcontrolador instalado, podendo variar de 1024 a 4096 *bytes* (WIECHERT, 2012). Entretanto, o projeto requer uma quantidade de dados muito maior e, por essa razão, esse método não poderia ser utilizado. Mesmo assim, é possível armazenar dados no Arduino em um cartão SD. Para isso, é preciso utilizar um módulo para cartão SD, necessitando que a comunicação seja feita com base no protocolo SPI (*Serial Peripheral Interface*), que consiste em uma comunicação serial síncrona de dados entre um *maste*r (mestre) e um *slave* (escravo), que neste caso são respectivamente, o Arduino e o *shield* para o cartão SD. Com o auxílio de uma biblioteca SD e a criação de um arquivo em formato “.txt”, pode-se armazenar dados e realizar a leitura deles com o Arduino (VELEDAF, 2014).

2.4. Projeto do *software*

Para a projeto do *software* é necessário desenvolver uma programação capaz de utilizar os dados obtidos pelo GPS e, a partir deles, determinar a localização do ônibus, calcular a distância até a próxima parada e estimar o tempo de viagem. Essas informações precisam ser constantemente atualizadas e informadas de maneira correta aos usuários. Para tornar essa tarefa possível, as ações a serem executadas pelo *hardware* serão descritas através de casos de uso, facilitando o desenvolvimento do *software*.

2.4.1. Cálculo da posição e estimativa de tempo até a próxima parada

A aquisição dos dados via GPS através de módulos ou *shields* já foi descrita na seção 2.3.2. Essa aquisição das coordenadas de latitude e longitude, além da data e horário, é feita de modo instantâneo.

Para o *shield* de GPS da Sparkfun aplicado ao Arduino, a biblioteca TinyGPSPlus desenvolvida para Arduino e o código da programação podem ser encontrados no site da própria empresa. Ao ser implementado no Arduino, o módulo fornece as coordenadas da latitude e longitude, data e horário em tempo real, na janela de mensagem da IDE do Arduino (CALVIS, 2015).

Para o Raspberry Pi, após conectar o módulo GPS através de uma entrada USB, é necessário configurar o adaptador USB e instalar o GPS Daemon, um pacote de *software* que pode ser baixado através de comandos aplicados no próprio Raspberry Pi e que auxilia a comunicação entre o módulo de GPS e o Raspberry Pi. Aplicando mais alguns comandos, é mostrado na tela de saída a data, o horário, a latitude e a longitude em tempo real (TOWNSEND, 2013).

As coordenadas de latitude e longitude são utilizadas no cálculo da posição atual do ônibus e a distância até a próxima parada. A data e o horário são úteis ao cálculo do tempo estimado até a próxima parada.

2.4.2. Gerenciamento de mensagens no monitor e no alto-falante

Para poder gerenciar as informações fornecidas ao usuário através de mensagens visuais no monitor ou sonoras no alto-falante é necessário, primeiramente, armazenar essas informações no *hardware,* como foi visto na seção 2.3.3.

Na seção 2.3.1, foi descrito a utilização de um MP3 *Player Shield* para Arduino capaz de tocar áudios em formato MP3. Esses áudios são armazenados previamente e o *software* precisa ser capaz de sincronizar o instante do anúncio sonoro com chegada à parada de destino. Para implementar o sistema é necessário instalar as bibliotecas SFEMP3 *Shield* e SdFat que auxiliam a comunicação do *shield* ao Arduino, e desenvolver um código que execute as tarefas desejadas (LABORATÓRIO DE GARAGEM, 2013). Com relação às mensagens mostradas no monitor, o *software* precisa ser capaz de analisar a localização do ônibus e mostrar no monitor a parada seguinte, de acordo com o itinerário que está armazenado no *hardware.*

No Raspberry Pi, a lógica utilizada no *software* é semelhante, entretanto, não será necessário a utilização de módulos ou *shields* para auxiliar o trabalho do *hardware.*

2.5. Escolha do *hardware*: Arduino x Raspberry Pi

Através das análises de *hardware* e *software,* percebe-se que para implementar o projeto no Arduino é necessário a utilização de módulos ou *shields* para diferentes funcionalidades como: anúncio de paradas em um monitor ou alto-falante, coleta de dados do GPS e armazenamento de itinerários e mensagens. No Raspberry Pi, o único módulo utilizado foi na aquisição de dados pelo GPS.

A comparação de modelos mais simples de Arduino com o Raspberry Pi pode ser injusta devido ao fato dessas placas não possuírem uma capacidade de processamento elevado ou presença de entradas que desempenhem funções variadas. A versão mais potente do Arduino levando em consideração a capacidade de processamento é o Galileo, que possui um processador Intel de 400 MHz e 32 bits (ainda inferior ao Raspberry Pi), e entrada para cartão micro SD (ARDUINO, 2014). Mesmo assim, ainda não existem saídas para áudio ou monitor, o que traz novamente a necessidade de utilizar *shields* capazes de exercer essas atividades.

A maior capacidade de processamento e a possibilidade de dispensar o uso de diversos *shields* tornam o Raspberry Pi um *hardware* mais prático e vantajoso que o Arduino para o desenvolvimento do projeto proposto.

2.6. Algoritmo de GPS: cálculo da distância e do tempo de viagem

Segundo MEZA, LIZÁRRAGA e LA FUENTE (2013) é possível estimar a distância e o tempo de viagem a partir dos dados coletados com o GPS. A metodologia utilizada para alcançar este propósito possui duas análises diferentes. A primeira consiste na análise dos sinais obtidos por GPS que

fornecerá as estatísticas das rotas. Obtidas essas informações, será realizada uma análise dos segmentos de rua. Essa metodologia pode ser observada no diagrama de atividades da Figura 3.

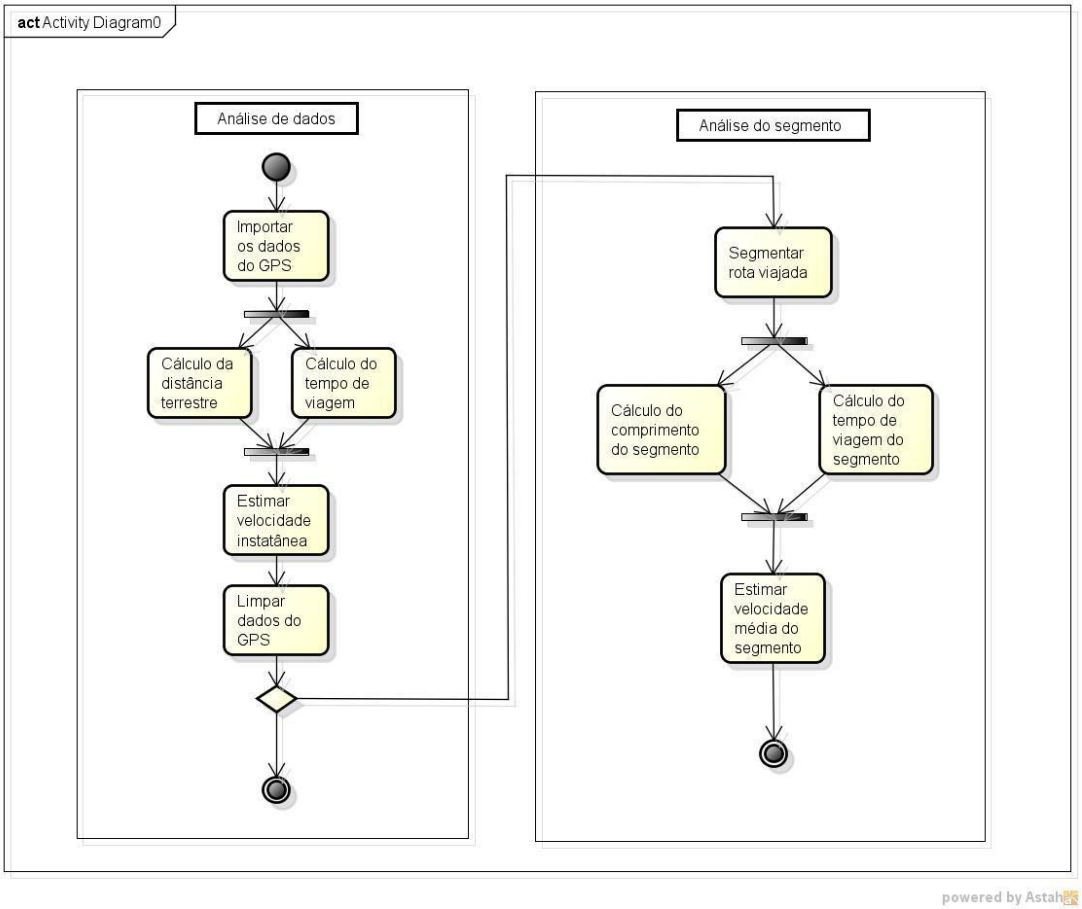

Figura 3 – Metodologia utilizada no cálculo da distância e do tempo de viagem a partir dos dados do GPS (MEZA, LIZÁRRAGA e LA FUENTE, 2013).

A sequência de ações dessa metodologia é:

1. Importar os dados obtidos por GPS: data, horário, latitude e longitude em tempo real.

2. Calcular a distância terrestre entre 2 coordenadas consecutivas.

Para isso é necessário calcular a distância entre 2 latitudes e 2 longitudes consecutivas. A partir de uma visão geométrica, cada uma dessas distâncias corresponde aos catetos de um triângulo retângulo e a hipotenusa corresponde à distância terrestre. O equacionamento fica:

$$A = 69{,}1 * (lat2 - lat1)$$

$$B = 69{,}1 * (lon2 - lon1) * cos(\mathrm{l}at)$$

$$d = \sqrt{A^2 + B^2} * 1609{,}334$$

Onde A e B são os catetos, $lat2$ e $lat1$ são as coordenadas da latitude, $lon2$ e $lon1$ são as coordenadas da longitude. As constantes 69,1 e 57,3 servem para converter as coordenadas da latitude e longitude em graus para distância terrestre em milhas. A distância terrestre d é convertida para metros multiplicando por $1609{,}334$.

3. Calcular o tempo de viagem instantâneo entre 2 coordenadas consecutivas dado por:

$$t = (ct2 - ct1) * 86400$$

Onde t é o tempo de viagem em segundos entre as duas coordenadas de GPS, com $ct2$ e $ct1$ sendo o tempo registrado em cada coordenada. Como esse tempo é dado em horas, multiplica-se por 86400 para convertê-lo em segundos.

4. Calcular a velocidade instantânea entre 2 coordenadas consecutivas, dado por:

$$v = \frac{d}{t} * 3{,}6$$

Onde é a velocidade instantânea em km/h, d é a distância em metros e t é o tempo em segundos. Para converter de m/s para km/h, multiplica-se por 3,6.

5. Limpar dados do GPS devido à erros de medição que podem ocorrer em áreas urbanas nas quais a cobertura dos sinais de satélites são enfraquecidas por prédios, túneis, condições climáticas, entre outros.

6. Segmentar as ruas em trechos obtidos por cada aquisição de dados do GPS ou trechos de comprimento constante, como por exemplo, 500 m ou 1 km.

7. Calcular o comprimento do segmento dado por:

$$L = \sum_{i=1}^{n} d$$

Onde n é o número de trechos e d é a distância entre 2 pontos consecutivos do GPS que formam cada trecho.

8. Calcular o tempo de viagem do segmento dado por:

$$T = \sum_{i=1}^{n} t$$

Onde n é o número de trechos e d é o tempo entre 2 pontos consecutivos do GPS que formam cada trecho.

9. Calcular a velocidade média do segmento dado por:

$$V = \frac{L}{T} * 3{,}6$$

Onde L é o comprimento do segmento e T é o tempo de viagem do segmento. Multiplica-se o resultado por 3,6 para obter a velocidade em km/h.

Um exemplo de estudo de caso baseado nessa metodologia foi realizado em Pequim. Os dados foram obtidos pelo Microsoft Research Group através da análise de taxis com GPS embutido, durante um período de 8 dias. Esses dados correspondem à um percurso de 90 minutos, das 1:30 PM às 3:00 PM. Um pedaço da análise dos segmentos de rua encontra-se na Tabela 2.

Tabela 2 – Análise dos segmentos de rua com o comprimento do segmento, o tempo de viagem e a velocidade média (MEZA, LIZÁRRAGA e LA FUENTE, 2013).

Seg ID	Date - Time	Seg Length (m)	Travel Time (s)	AV Speed (km/h)
1	01:32:34 p.m.	405	105.00	14
2	01:33:39 p.m.	427	64.00	24
3	01:36:00 p.m.	818	141.00	18
4	01:36:45 p.m.	285	45.00	23
5	01:37:50 p.m.	497	65.00	28
6	01:38:30 p.m.	439	40.00	40
7	01:39:15 p.m.	532	45.00	42
8	01:39:45 p.m.	179	30.00	21
9	01:41:50 p.m.	426	95.00	16
10	01:42:15 p.m.	34	25.00	5
11	01:43:50 p.m.	523	105.00	19
12	01:51:25 p.m.	1018	455.00	7
13	01:53:57 p.m.	475	152.00	12
14	01:56:02 p.m.	995	125.00	27
15	01:58:47 p.m.	1291	165.00	29
16	01:59:32 p.m.	462	45.00	36

17	01:59:52 p.m.	191	20.00	19
18	02:05:12 p.m.	1099	320.00	10
19	02:07:57 p.m.	296	165.00	7
20	02:08:47 p.m.	552	50.00	41

Os parâmetros de um trajeto percorrido por um veículo podem ser estimados através da análise dos segmentos intermediários e quanto menor o segmento, maior será a precisão do cálculo. Percebe-se pela análise da tabela 2 que os parâmetros comprimento do segmento, tempo de viagem e velocidade média são calculados sempre a partir de 2 coordenadas consecutivas do GPS. Para obter a distância percorrida e o tempo de viagem para o trecho desejado, deve-se somar todos os valores obtidos nos segmentos até o ponto final. Por exemplo, do segmento 1 ao 10, a distância percorrida foi de 4042 m e o tempo de viagem foi de 655 s.

A partir dos resultados obtidos por esse estudo, conclui-se que é possível estimar, a partir de alguns dados obtidos por GPS, parâmetros como distância percorrida e tempo de viagem. Para que o método utilizado seja preciso, é necessária a utilização de um GPS com uma taxa de aquisição de dados adequada, diminuindo, desse modo, o intervalo de tempo entre cada aquisição.

Entretanto, esse método não possibilita a realização de estimativas em tempo real, já que seus cálculos são obtidos a partir de um banco de dados, coletado através do deslocamento de veículos equipados com GPS.

3. Descrição do Sistema Proposto

Neste capítulo, é realizada a descrição do sistema de anúncio de paradas nas linhas de ônibus. Para auxiliar na descrição do funcionamento do sistema são utilizados diagramas UML, que mostram como é a interação do sistema com o usuário, a sequência de eventos e a estrutura utilizada no desenvolvimento do *software*. Nesse projeto, são utilizados 4 tipos de diagramas UML: diagrama de casos de uso, diagrama de atividades, diagrama de classes e diagrama de sequência.

3.1. Diagrama de Casos de Uso

Os diagramas de casos de uso são muito úteis na descrição de como o usuário interage com o sistema e quais as funcionalidades envolvidas. Nesse projeto, o usuário não executa nenhum caso de uso, já que a maior parte das ações é executada pelo sistema. Há apenas a presença de um operador que é responsável por selecionar o itinerário e iniciar/finalizar a operação da plataforma. O diagrama de casos de uso do sistema proposto pode ser observado na Figura 4.

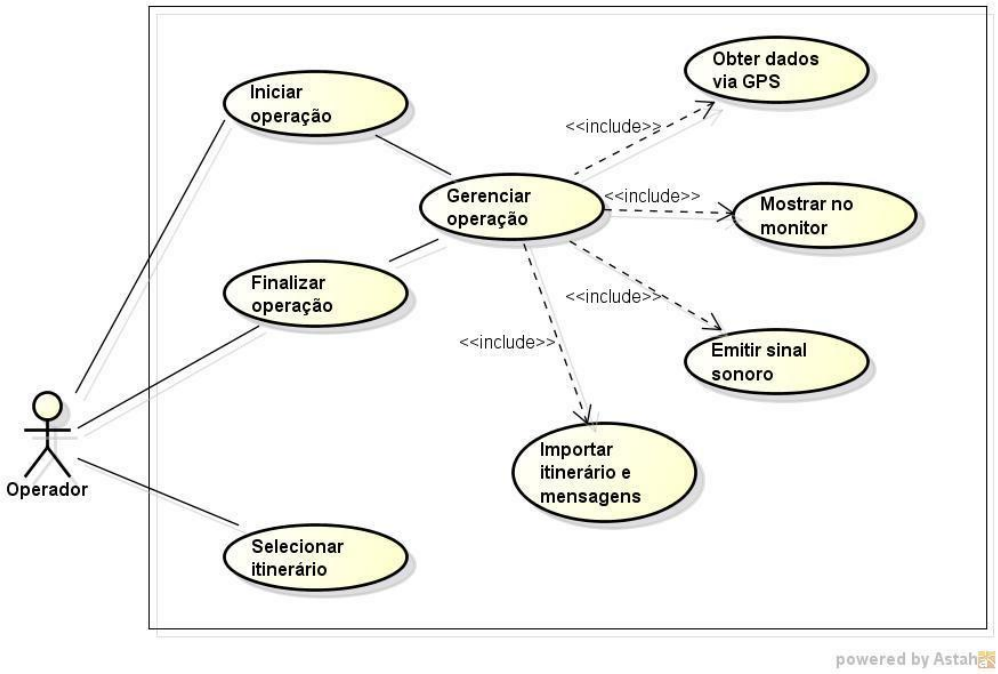

Figura 4 – Diagrama de casos de uso do sistema proposto

A seguir será realizada uma descrição detalhada dos casos de uso e como eles estão relacionados com os demais casos de uso do sistema.

- **Iniciar operação:** O operador pode iniciar a operação do sistema. Uma vez que o itinerário esteja selecionado, inicia-se o gerenciamento da operação.

- **Finalizar operação:** De maneira análoga ao caso de uso anterior, o operador pode finalizar a operação do sistema para realizar ajustes, consertar erros e reconfigurar a plataforma.

- **Gerenciar operação:** Este caso de uso é a base de todo o funcionamento do sistema. A plataforma será responsável pelo gerenciamento de todas as operações presentes e deve realizar outros 4 casos de uso: '**Obter dados via GPS**', '**Mostrar no monitor**', '**Emitir sinal sonoro**' e '**Importar itinerário e mensagens**'. Esses 4 casos de uso possuem uma relação de inclusão com o caso de uso '**Gerenciar operação**' e esta relação está representada no diagrama com a palavra-chave <<include>>. Além disso, a plataforma possui também relação de associação com os casos de uso '**Iniciar operação**' e '**Finalizar operação**'.

- **Obter dados via GPS:** Para saber a localização do ônibus, calcular a distância até a próxima parada e o tempo de viagem, é preciso realizar uma aquisição das coordenadas via GPS. Durante esse processo, a plataforma obtém a data, o horário, a latitude e a longitude em tempo real. A velocidade com que esse processo é feito depende da taxa de aquisição do GPS e isso acaba influenciando os cálculos realizados no *software*.

- **Mostrar no monitor:** Os usuários do ônibus são informados de maneira visual por meio de um monitor LCD que contém as informações das próximas paradas. Após passar por uma determinada parada, o monitor atualiza e passa a mostrar as informações da próxima parada.

- **Emitir sinal sonoro:** Caso de uso que é responsável pela emissão das mensagens de aviso aos usuários do ônibus, assim que este estiver próximo da parada seguinte. Sendo assim, esse caso de uso depende do tempo de viagem e sua ativação ocorre quando este parâmetro se aproxima de um valor pré-determinado.

- **Importar itinerário e mensagens:** Caso de uso em que a plataforma recebe o itinerário que contém as informações sobre a linha de ônibus e as mensagens de voz que fazem o anúncio das paradas aos usuários. Essas informações (em especial o itinerário) são essenciais ao funcionamento do sistema, uma vez que o cálculo de diversos parâmetros é dependente delas.

- **Selecionar itinerário:** O operador é responsável por selecionar o itinerário correto que determina o caminho percorrido pelo ônibus

3.2. Diagrama de Atividades

Com o objetivo obter um maior aprofundamento das atividades executadas pela plataforma, utilizou-se o diagrama de atividades para obter uma descrição detalhada dos casos de uso que ocorrem durante o funcionamento da plataforma e que são executadas pelo operador.

3.2.1. Gerenciar operação

O diagrama de atividades que ilustra o gerenciamento das operações executadas pela plataforma pode ser observado na Figura 5. O diagrama mostra que a primeira atividade executada pela plataforma é importar o itinerário do ônibus e as mensagens informativas aos usuários mostradas no monitor. A seguir, é feito a leitura das coordenadas obtidas pelo GPS. A partir delas e com as coordenadas da próxima parada armazenadas, é calculada a velocidade do ônibus e posteriormente o tempo de viagem estimado. A partir desse ponto, há uma tomada de decisão que depende do quão próximo está o ônibus em relação à próxima parada. Assumiu-se que 3 minutos seja um tempo necessário e suficiente para que as pessoas possam se aproximar da porta, levando em consideração que o ônibus pode estar cheio, dificultando a locomoção até a saída. Enquanto o tempo de viagem for maior do que 3 minutos, a plataforma continua mostrando as próximas paradas no monitor. Caso o tempo de viagem fique menor do que 3 minutos, a plataforma identifica que o ônibus está se aproximando da próxima parada e com isso, a próxima parada é enfatizada no monitor LCD e é emitido um sinal sonoro alertando os usuários da chegada à parada seguinte.

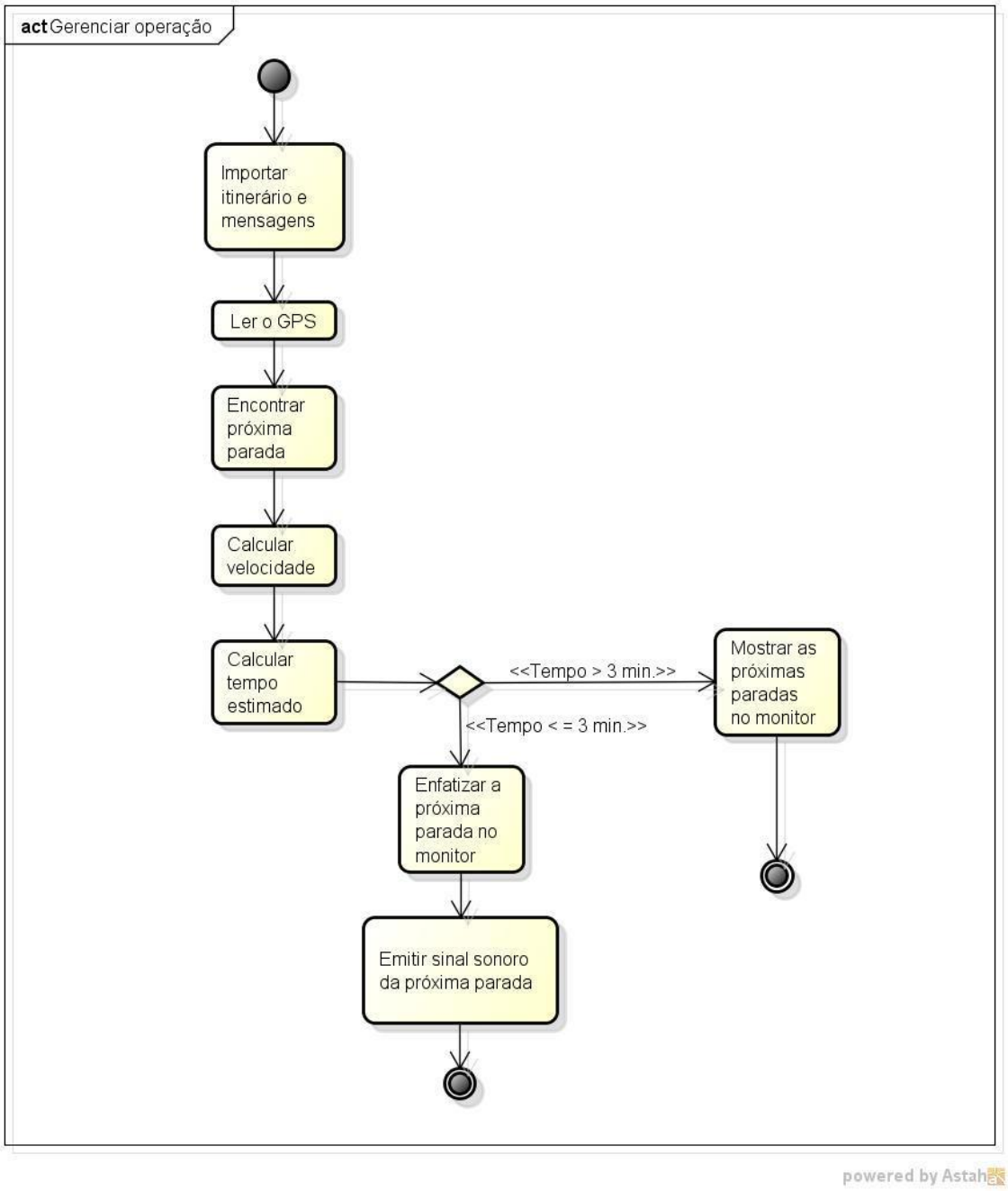

Figura 5 – Diagrama de atividades – Gerenciar operação

3.2.2. Selecionar itinerário

O diagrama de atividades que ilustra a seleção do itinerário executado pelo operador pode ser observado na Figura 6. O diagrama mostra que a primeira atividade executada pelo operador é inserir o cartão de memória na plataforma. A seguir, liga-se a plataforma e dentre os possíveis itinerários, é selecionado um deles. Por fim, é feito uma confirmação da seleção.

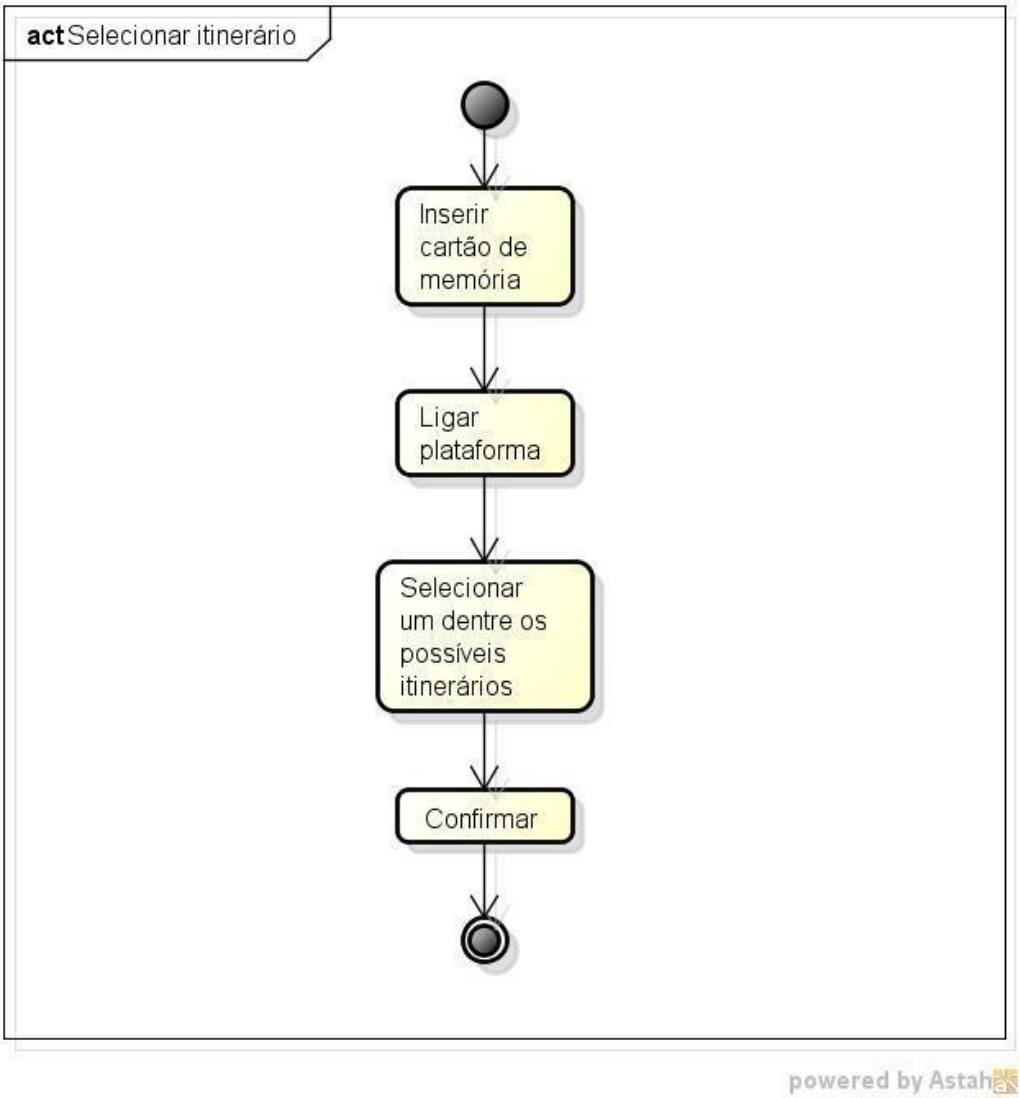

Figura 6 – Diagrama de atividades – Selecionar itinerário

3.3. Diagrama de Classes

O diagrama de classes é essencial na caracterização da estrutura do *software*. Esse tipo de diagrama mostra o tipo de relação entre diferentes classes, além dos atributos presentes e métodos utilizados. Na Figura 7, encontra-se o diagrama de classes que mostra como está organizado o *software*, além das principais funções utilizadas no gerenciamento de operações.

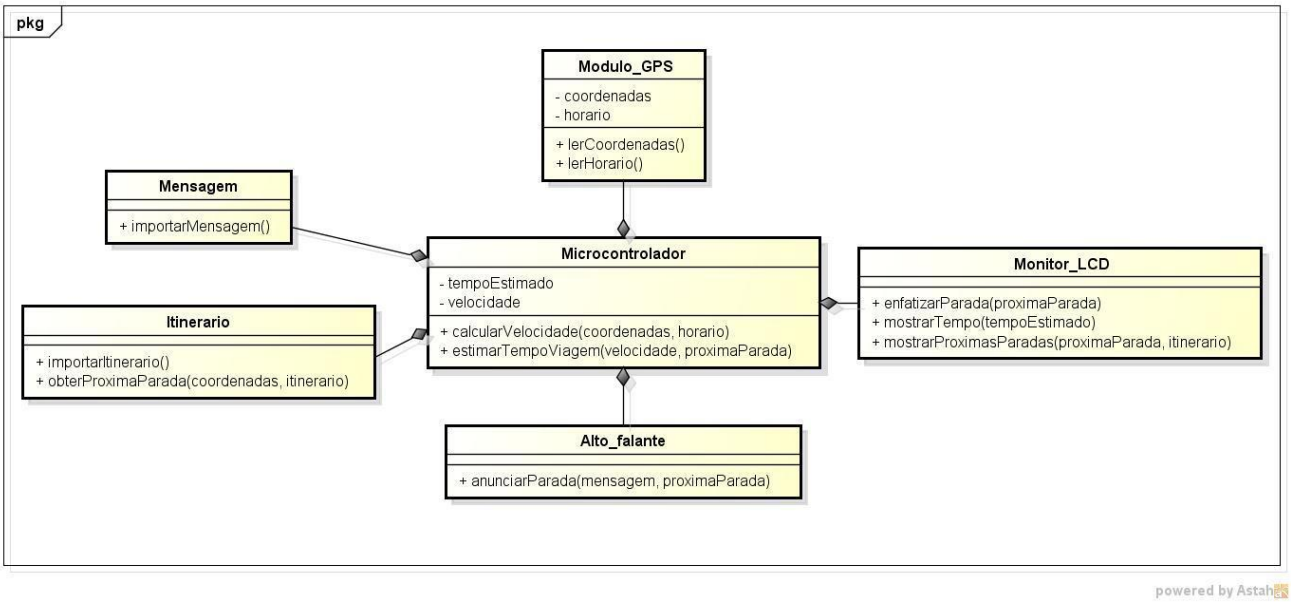

Figura 7 – Diagrama de classes – Gerenciamento de operações do sistema

A seguir, é feita uma descrição detalhada das classes. Observa-se que há uma relação de composição entre o microcontrolador e as outras classes. Além disso, o microcontrolador é a classe mais importante desse sistema e é responsável pelo gerenciamento de todas as outras classes presentes. Todos os métodos utilizados pelo sistema serão chamados pela classe '**Microcontrolador'**.

3.3.1. Mensagem

Classe que representa as mensagens de voz utilizadas no sistema. Possui um atributo 'mensagem', que armazena as mensagens na plataforma, essas mensagens são obtidas pelo método 'importarMensagem().

3.3.2. Itinerário

Classe que representa os itinerários utilizados no sistema. Possui um atributo 'itinerario' que armazena o itinerário utilizado na plataforma. Além disso, possui os seguintes métodos:

- 'importarItinerario()': obtém o itinerário que será utilizado e armazena em 'itinerario'.

- ‘obterProximaParada(coordenadas,itinerario)’: utiliza os parâmetros ‘coordenadas’, obtida pelo GPS, e ‘itinerario’, baseado no itinerário proposto, obtendo-se a próxima parada.

3.3.3. Modulo_GPS

Classe que representa o módulo de GPS utilizado na aquisição de dados e coordenadas. Possui 2 atributos: ‘coordenadas’ que representa as coordenadas de latitude e longitude e ‘horario’ que representa o horário no instante da aquisição do GPS. Possui os seguintes métodos:

- ‘lerCoordenadas()’: a plataforma realiza a leitura das coordenadas obtidas pelo módulo GPS.

- ‘lerHorario()’: a plataforma realiza a leitura do horário obtido pelo módulo GPS.

3.3.4. Monitor_LCD

Classe que representa o monitor de LCD utilizado para mostrar informações aos usuários de maneira visual. Não possui atributos. Possui os seguintes métodos:

- ‘enfatizarParada(proximaParada)’: utiliza o parâmetro ‘proximaParada’ e com isso a plataforma mostra no monitor de LCD de maneira contundente, a aproximação da parada seguinte.

- ‘mostrarTempo(tempoEstimado)’: utiliza o parâmetro ‘tempoEstimado’ e com isso a plataforma mostra no monitor de LCD o tempo de viagem estimado.

- ‘mostrarProximasParadas(proximaParada,itinerario)’: utiliza os parâmetros ‘proximasParadas’ e ‘itinerario’, e com isso a plataforma mostra no monitor de LCD as 4 paradas seguintes.

3.3.5. Alto_falante

Classe que representa o alto-falante utilizado no anúncio de informações aos usuários de maneira sonora. Não possui atributos. Possui o método ‘anunciarParada(mensagem, proximaParada)’ que depende dos parâmetros ‘mensagem’ e ‘proximaParada’, e com isso a plataforma anuncia através do alto-falante a aproximação da parada seguinte.

3.3.6. Microcontrolador

Classe que representa o microcontrolador utilizado no sistema. Possui 2 atributos: ‘tempoEstimado’ que representa o tempo de viagem estimado e ‘velocidade’ que representa a velocidade desenvolvida pelo ônibus. Possui os seguintes métodos:

- ‘calcularVelocidade(coordenadas,horario)’: utiliza os parâmetros ‘coordenadas’ e ‘horario’ e retorna a velocidade desenvolvida pelo ônibus.

- ‘estimarTempoViagem(velocidade,proximaParada)’: utiliza os parâmetros ‘velocidade’ e ‘proximaParada’ e retorna o tempo de viagem estimado.

3.4. Diagrama de Sequência

O diagrama de sequência representa as interações entre os objetos do sistema projetado e a ordenação temporal dos processos que serão realizados no programa, de forma lógica e simples.

No sistema projetado tem-se a centralização dos processos no Microcontrolador, uma vez que este é o responsável pelo caso de uso ‘Gerenciar Operação’, conforme mostrado na Figura 8.

Primeiramente, o programa principal importa o itinerário que será percorrido por meio da chamada do método ‘importarItinerário()’ da classe ‘**Itinerário**’. Feito isso, o sistema chama os métodos ‘lerCoordenadas()’ e ‘lerHorário()’ da classe ‘**Módulo_GPS**’. A partir da sua localização geográfica e do itinerário obtidos anteriormente, obtém a próxima parada chamando o método ‘obterProximaParada(coordenadas,itinerario)’. Com os parâmetros obtidos, são chamados os métodos ‘calcularVelocidade(coordenadas,horario)’ e ‘estimarTempoViagem(velocidade, proximaParada)’ da classe ‘**Microcontrolador**’, a fim de calcular a velocidade instantânea do ônibus e o tempo estimado até a próxima parada.

Terminadas todas essas operações, se o tempo estimado for superior a 3 minutos, o sistema mostra os resultados obtidos visualmente (Figura 9) por meio chamada dos métodos ‘mostrarTempo(tempoEstimado)’ e ‘mostrarPróximasParadas(proximaParada,itinerario)’ da classe ‘**Monitor_LCD**’. Caso contrário (tempo estimado inferior a 3 minutos), importa a mensagem sonora relacionada a próxima parada e faz o seu anúncio por meio do alto-falante chamando os métodos ‘importarMensagem()’ da classe ‘**Mensagem**’ e ‘anunciarParada(mensagem, proximaParada)’ da classe ‘**Alto-falante**’, além de enfatizar de forma visual a próxima parada chamando o método ‘enfatizarParada (proximaParada, itinerario)’, conforme visto na Figura 10.

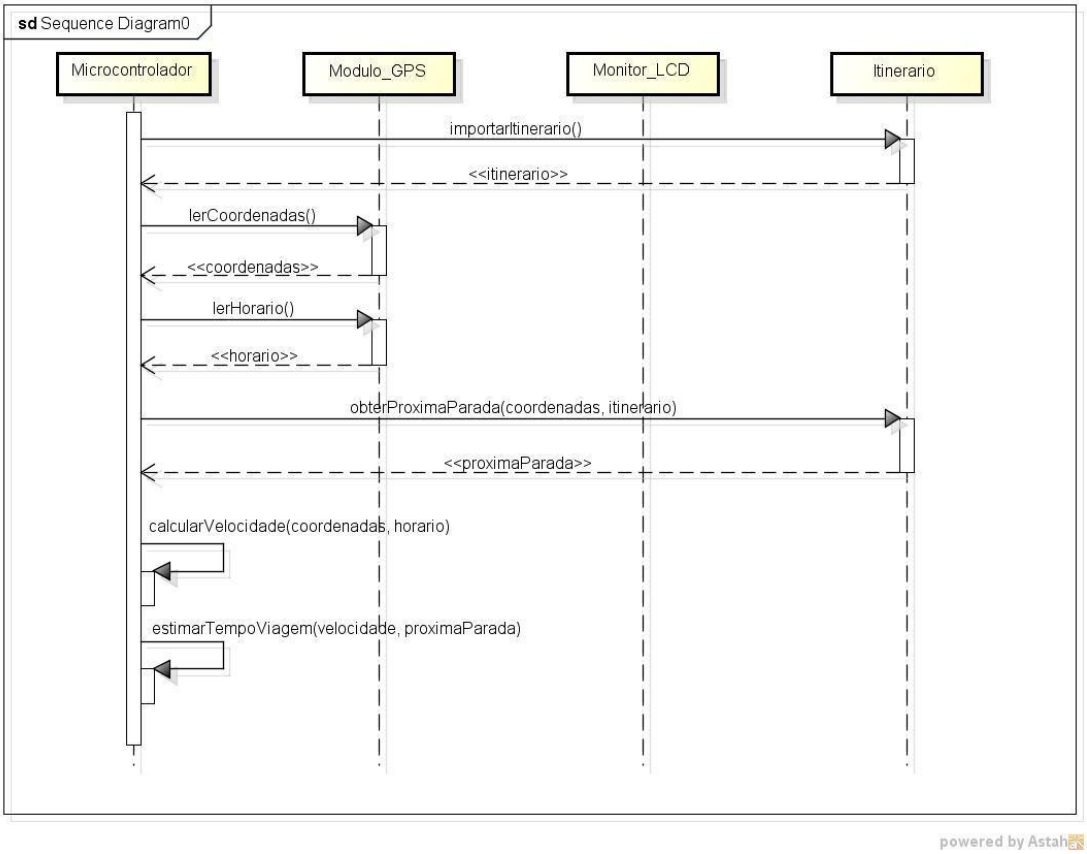

Figura 8 – Diagrama de sequência - Gerenciar Operação

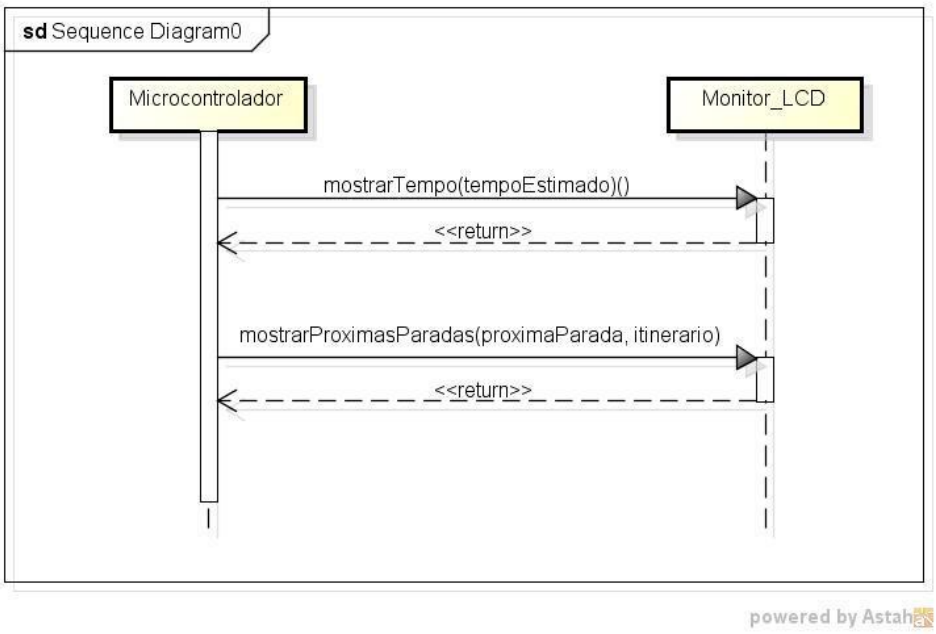

Figura 9 – Diagrama de sequência para tempo > 3 min.

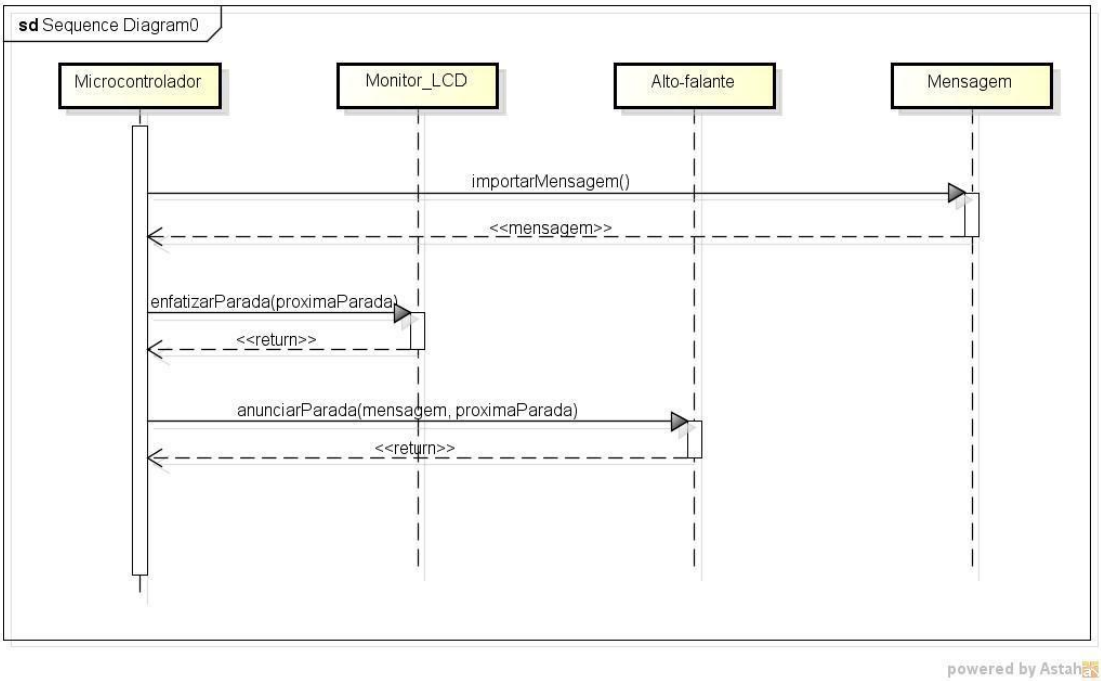

Figura 10 – Diagrama de sequência para tempo < 3 min.

4. Teste do Sistema de Aquisição de Dados via GPS

Para realizar o teste do sistema de aquisição de dados via GPS, utilizou-se o Arduino e o módulo GPS da Adafruit. Esse sistema proporcionará maior praticidade durante o teste, já que o Arduino envia as informações diretamente à tela do *notebook*, além deste já estar alimentando o Arduino. Com isso, o teste do sistema pode ser feito em movimento, dentro de um carro ou no próprio ônibus.

Entretanto, o objetivo desse projeto é desenvolver um sistema embarcado que possa ser instalado dentro de um ônibus. O Arduino não fornece essa possibilidade pois necessita do acompanhamento de um computador. Para a implementação final do projeto, será utilizado o Raspberry Pi que possui características que atendem essa finalidade.

4.1. Instalação do Módulo GPS no Arduino

A instalação do módulo GPS no Arduino é relativamente simples. O módulo possui 9 pinos, mas foram utilizados apenas 4. Os pinos utilizados são VIN, GND, RX e TX. O pino VIN corresponde à alimentação de 3 a 5 V. O pino GND é o sinal de terra do módulo. O pino TX transmite os dados do módulo GPS ao microcontrolador e funciona ao nível lógico 3,3 V. O pino RX é responsável por enviar dados ao GPS e funciona tanto a 3,3 V quanto a 5 V. Os pinos TX e RX funcionam a uma taxa de baud padrão de 9600 Os pinos VIN e GND serão conectados ao 5 V e ao terra (GND) do Arduino respectivamente. Os pinos RX e TX foram conectados a dois pinos de entrada digital do Arduino (ADAFRUIT, 2012). A conexão dos pinos pode ser observada na Figura 11.

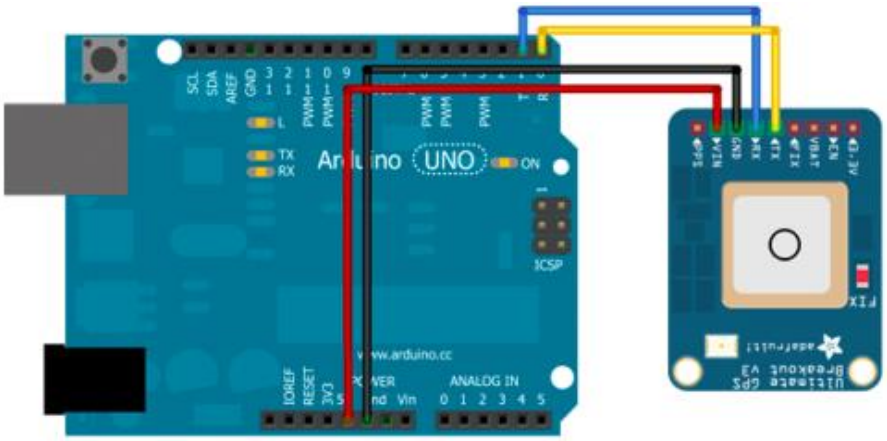

Figura 11 – Esquemático das conexões com o módulo GPS (ROBOTIC-CONTROLS, 2013)

Com o intuito de aumentar a captação do sinal de GPS, foi utilizada uma antena externa que pode ser conectada ao módulo GPS. Essa antena proporciona um ganho maior ao sistema, mas também necessita de uma quantidade maior de corrente. Além disso, ela possui um imã em seu interior, facilitando a fixação em superfícies metálicas (ADAFRUIT, 2012). A montagem desse sistema pode ser observada na Figura 12.

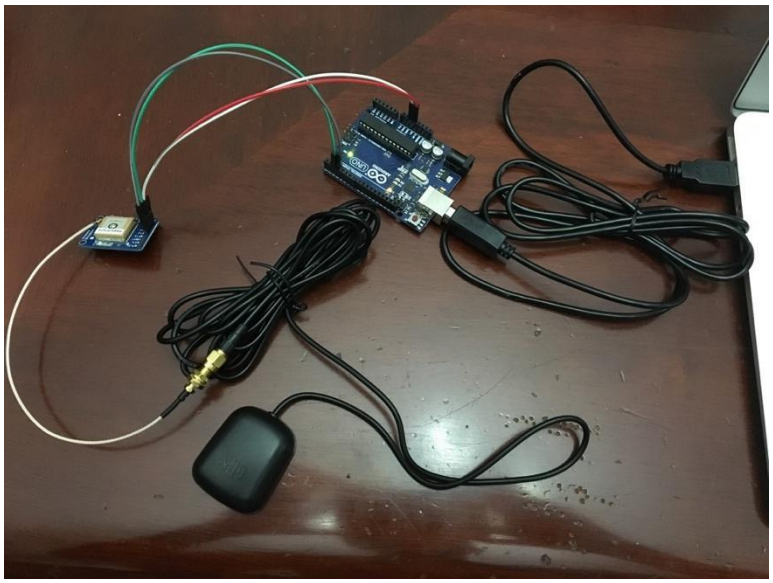

Figura 12 – Montagem do sistema composto pelo módulo GPS, antena externa e Arduino

4.2. Teste do Sistema de Aquisição via GPS

Para realizar o teste de aquisição de dados do GPS é necessário baixar a biblioteca Adafruit GPS. Essa biblioteca já possui exemplos de *sketch* para o Arduino e neles já são mostradas as coordenadas do GPS. Os valores são mostrados no monitor serial do *software* do Arduino e taxa de baud utilizada precisa ser 115200 (ADAFRUIT, 2012). Um exemplo de teste e os valores da aquisição de dados pode ser observado no Figura 13.

```
Time: 1:59:7.0
Date: 14/9/2016
Fix: 1 quality: 1
Location: 2336.2744S, 4638.0209W
Location (in degrees, works with Google Maps): -23.6046, -46.6337
Speed (knots): 0.14
Angle: 313.14
Altitude: 706.00
Satellites: 7
$PGTOP,11,3*6F
$GPGGA,015908.000,2336.2743,S,04638.0209,W,1,07,1.18,706.0,M,-3.5,M,,*77
$GPRMC,015908.000,A,2336.2743,S,04638.0209,W,0.14,309.55,140916,,,A*64
$PGTOP,11,3*6F
$GPGGA,015909.000,2336.2743,S,04638.0209,W,1,07,1.18,706.0,M,-3.5,M,,*76
$GPRMC,015909.000,A,2336.2743,S,04638.0209,W,0.12,263.13,140916,,,A*6C
```

Figura 13 – Monitor serial com a aquisição de dados via GPS

Pela análise desta figura percebe-se que diversos parâmetros são gerados: o horário e a data em que foi feita a aquisição, as coordenadas de latitude e longitude, a velocidade, o ângulo formado pelo módulo, a altitude e o número de satélites.

5. Implementação do Sistema

Para implementar o sistema, utilizou-se o ambiente de desenvolvimento Microsoft *Visual Studio 2015*, principal ferramenta que auxilia no desenvolvimento de aplicações baseadas em Windows IoT. Neste capítulo será descrito alguns aspectos gerais do sistema e o processo utilizado para implementá-lo.

5.1. Aspectos Gerais

Internet of Things é uma conceito na qual os dispositivos e processos estão se tornando a cada dia mais inteligentes e dinâmicos. O formato de sua arquitetura atende os requisitos e tecnologias solicitados pelos desenvolvedores e os possibilita solucionar problemas reais. Até o ano de 2020, 25 bilhões de dispositivos estarão conectados à internet, facilitando a utilização de dados e tomadas de decisões de maneira autônoma. A arquitetura IoT possui um escopo diversificado e pode contemplar tanto dispositivos físicos quanto virtuais, sensores, atuadores e diferentes protocolos (Figura 14) (RAY, 2016).

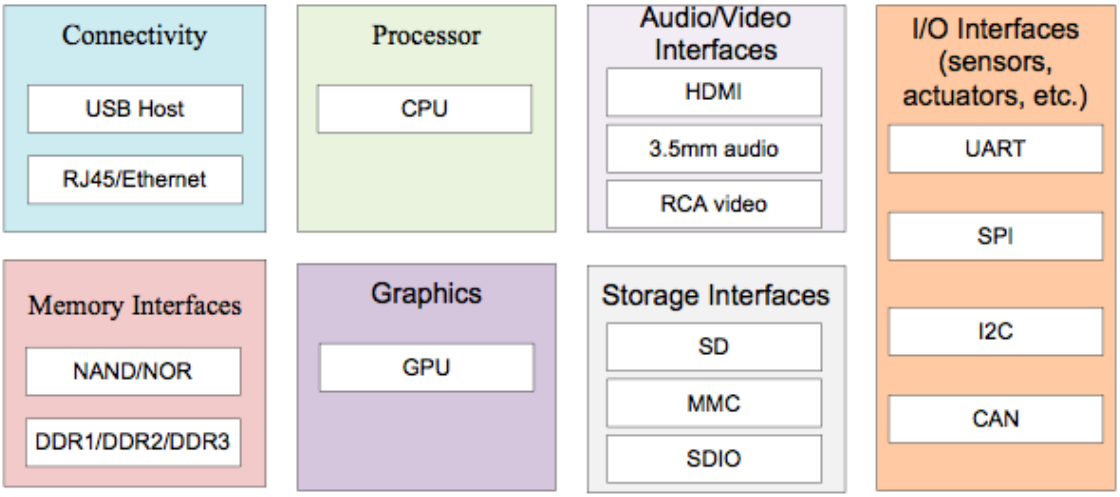

Figura 14 – Dispositivos e protocolos que funcionam via IoT (RAY, 2016).

Nesse projeto, essa funcionalidade é implementada utilizando o Windows IoT, uma versão de Windows 10 otimizado para plataformas menores como o Raspberry Pi. Essa ferramenta utiliza a Plataforma Universal Windows (UWP), facilitando o desenvolvimento de aplicações que possibilitam integrar interface com o usuário, pesquisa, armazenamento online e serviços baseado em nuvem (MICROSOFT, 2016).

5.2. Configuração e Implementação da plataforma

Antes de iniciar a implementação do sistema, é necessário realizar a instalação e configuração baseado no *hardware* (Raspberry Pi), na mídia (cartão MicroSD) e no sistema operacional utilizado (*Windows IoT Core)*. Para facilitar a configuração do dispositivo, pode-se usar o painel do *Windows IoT Core* (Figura 15). Durante a configuração, o sistema operacional foi instalado no cartão MicroSD e o dispositivo utilizado recebe um nome e uma senha de acesso. O ambiente de desenvolvimento utilizado foi o *Visual Studio 2015* que proporcionou *templates*, editor de código, debugador e diversas outras ferramentas que auxiliaram no desenvolvimento do projeto (MICROSOFT, 2016).

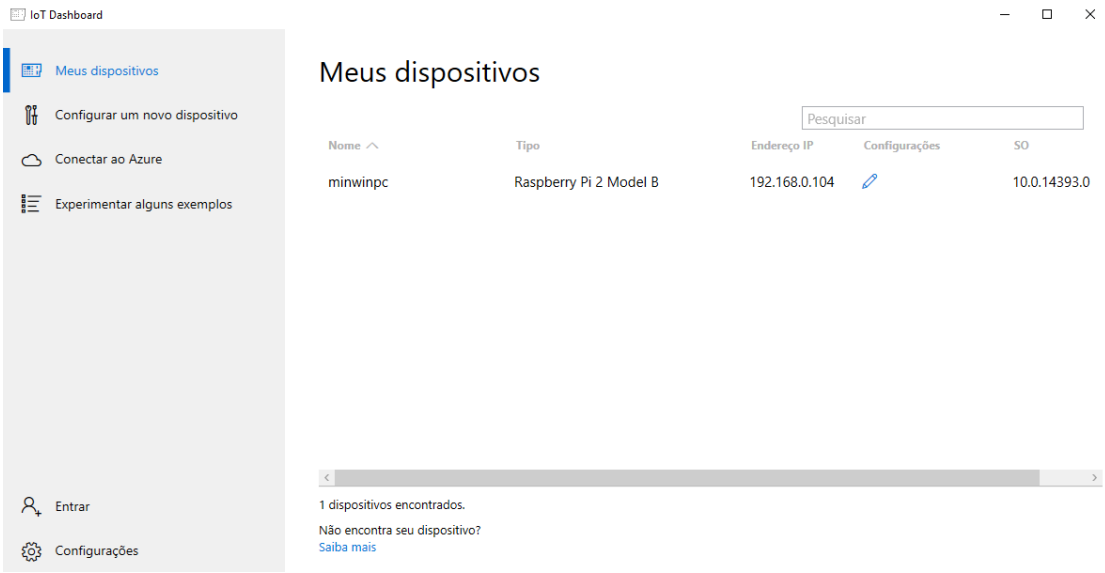

Figura 15 - Painel do *Windows IoT Core*

5.2.1. Montagem do Sistema

Para o sistema, foram utilizados um Raspberry Pi 2, um monitor LCD, um módulo GPS da Adafruit e uma caixa de som, e todos esses dispositivos foram integrados através de uma arquitetura IoT. A programação do *software* foi realizado em um *notebook* e transmitida ao Raspberry Pi via internet, sendo assim, é essencial que tanto o *notebook* quanto o Raspberry Pi estejam conectados à rede durante a implementação.

O monitor LCD foi conectado ao Raspberry Pi via HDMI ou VGA e é o dispositivo no qual serão mostradas as informações aos usuários do ônibus. Nele será projetado a interface gráfica na qual é possível acompanhar as próximas paradas, o tempo estimado de viagem e a velocidade do ônibus.

O Raspberry Pi 2 é o controlador do sistema e é responsável por enviar o conteúdo da interface ao monitor LCD, processar os dados do GPS e executar as rotinas presentes no *software*.

O módulo GPS é responsável por coletar diversos dados (latitude, longitude, velocidade, tempo e horário de aquisição) conforme mostrado na seção 4.3.

A caixa de som é responsável pelo anúncio das paradas de forma sonora. A idéia inicial era utilizar a própria saída HDMI para realizar essa tarefa, entretanto, verificou-se que essa funcionalidade não está disponível para ser implementada no Raspberry Pi 2 via *Windows Iot* (MICROSOFT, 2016). Sendo assim, foi utilizada a saída analógica de áudio de 3,5 mm.

O sistema implementado está representado na Figura 16. Observa-se que o *notebook* não está presente na imagem mas está conectado via wi-fi e o Raspberry Pi está conectado através de um cabo de rede. Não há nenhuma conexão física entre os dois dispositivos, o elo é estabelecido apenas pela arquitetura IoT. Vale ressaltar que uma vez que o *software* esteja carregado no Raspberry Pi, não é mais necessária a utilização do *notebook*.

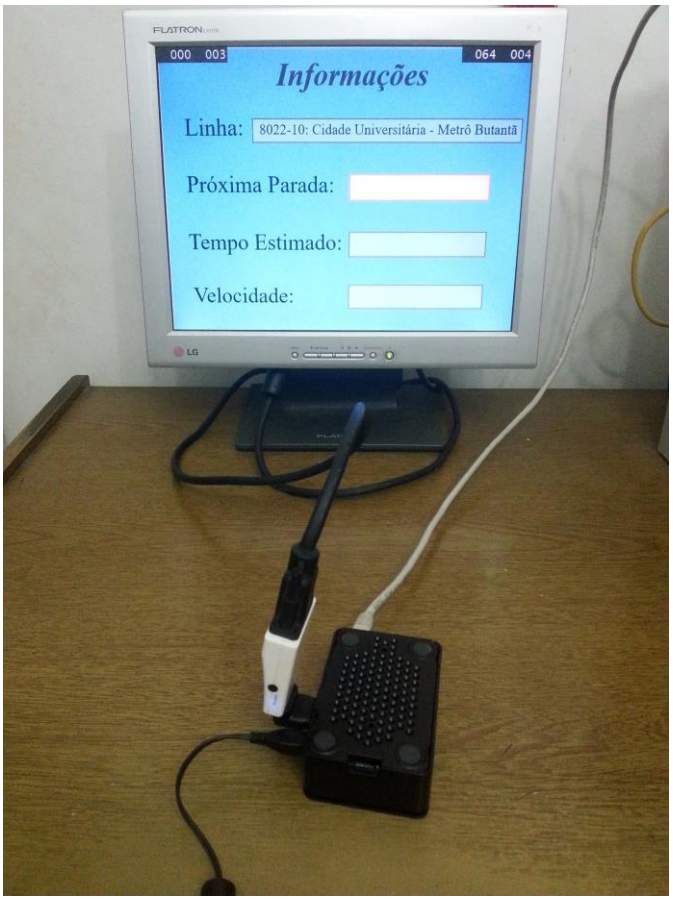

Figura 16 – Sistema implementado

5.2.2. Implementação da Interface Gráfica

A interface gráfica foi implementada utilizando o ambiente de desenvolvimento *Visual Studio 2015* (Figura 17) que oferece diversas ferramentas que auxiliam no *design*. Uma delas permite ao desenvolvedor arrastar os componentes desejados e posicioná-las com o *layout* desejado.

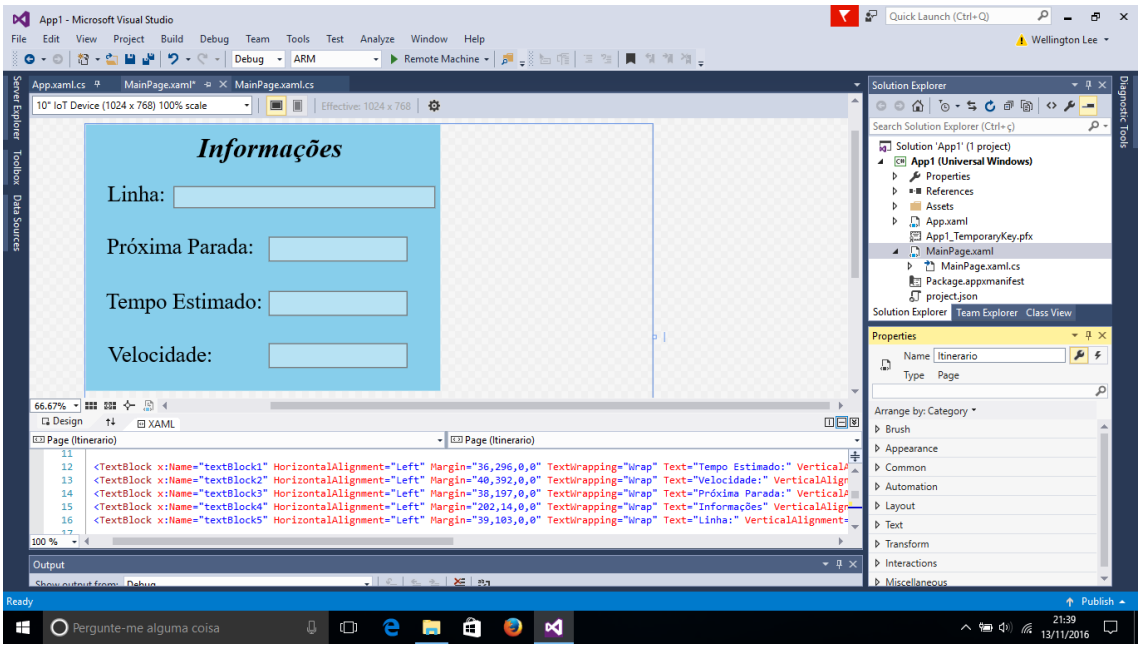

Figura 17 – Desenvolvimento da interface gráfica no *Visual Studio 2015*

Foram desenvolvidas 2 interfaces gráficas: uma para o operador do sistema e outra para o usuário (Figuras 18 e 19 respectivamente).

Na interface gráfica do operador, é possível escolher, dentre as linhas de ônibus disponíveis, qual será utilizada. Após a escolha do itinerário, o operador consegue iniciar a operação do sistema e com isso, a aquisição de dados via GPS começa de maneira simultânea. Esses dados serão utilizados para determinar a localização do ônibus e estimar o de viagem até a próxima parada. Há também a possibilidade de encerrar a operação em caso de necessidade.

A interface gráfica do usuário contém as informações úteis à viagem (linha de ônibus utilizada, próxima parada, tempo de viagem estimado e velocidade). Essas informações são calculadas pelo *software* implementado e são mostradas ao usuário no monitor LCD em tempo real.

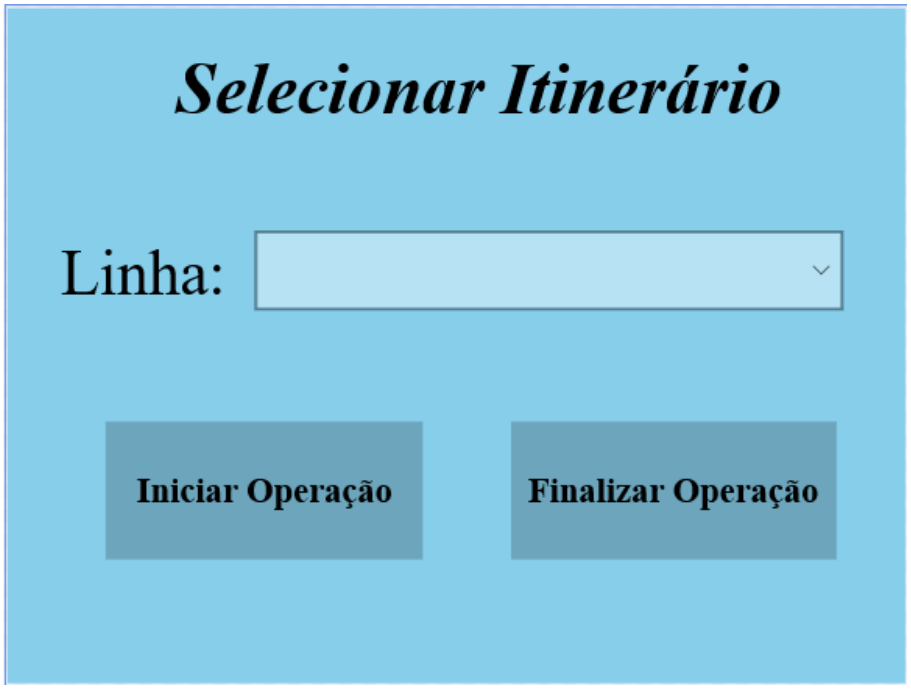

Figura 18 – Interface gráfica do operador

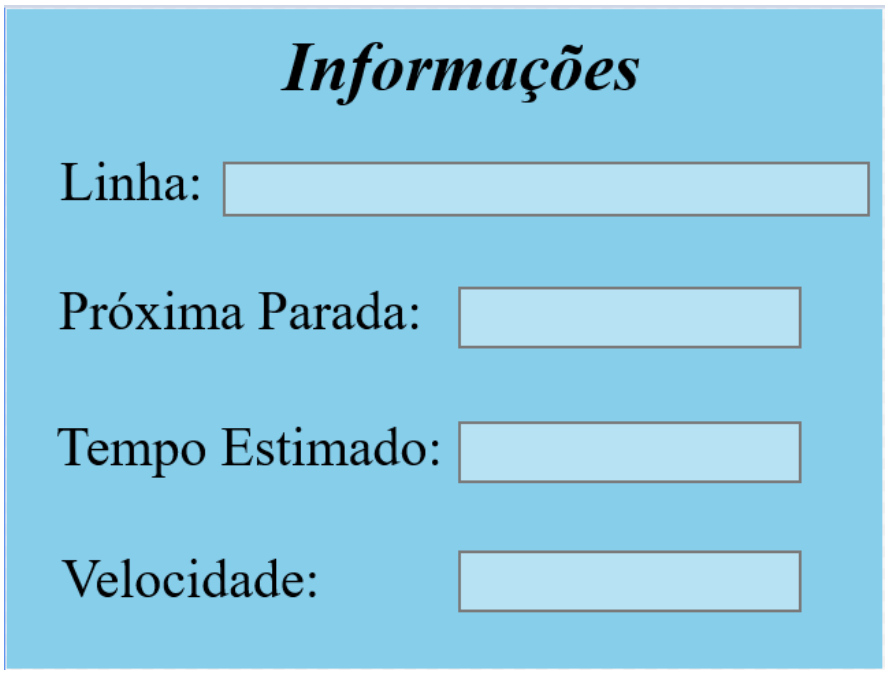

Figura 19 – Interface gráfica do usuário

5.2.3. Implementação do Algoritmo de Cálculo das Próximas Paradas e Tempo Estimado

Antes de iniciar a implementação do algoritmo, é necessário obter as paradas percorridas pelo ônibus e suas respectivas coordenadas de latitude e longitude. Os nomes de cada parada são mostradas no monitor LCD e as coordenadas são essenciais na localização e nos cálculos.

As coordenadas presentes nos pontos de ônibus foram obtidas via *Google Maps*. Para esse projeto, decidiu-se testar o itinerário do ônibus da Sptrans da Linha 8022-10. Esse ônibus parte do metrô Butantã, circula pela Cidade Universitária e retorna ao mesmo ponto de início (Figura 20). As coordenadas de latitude, longitude e a elevação em relação ao nível do mar podem ser observadas na Tabela 3.

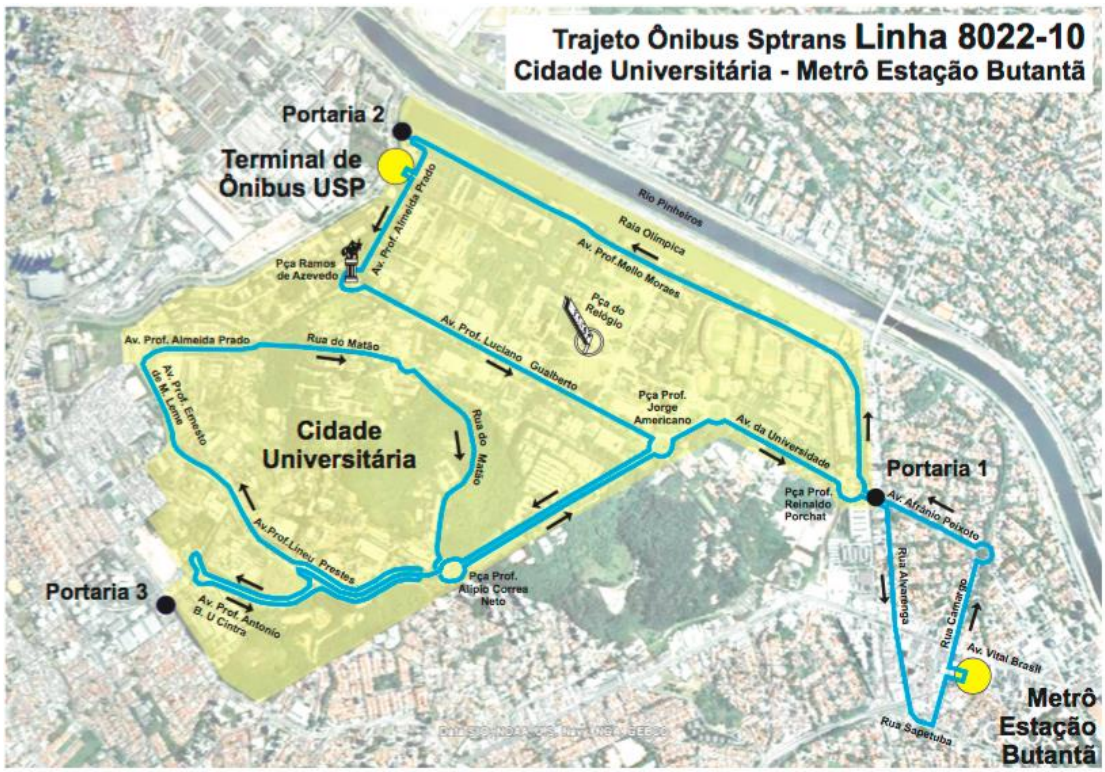

Figura 20 – Itinerário do ônibus da Sptrans da Linha 8022-10 (IME-USP)

Tabela 3 – Tabela que relaciona o ponto de ônibus com a respectiva latitude, longitude e elevação (MAPCOORDINATES e SPTRANS)

Ponto	Latitude	Longitude	Elevação (m)
Terminal Metrô Butantã	-23.57213697	-46.70893125	731
Av Afrânio Peixoto, 363	-23.56676157	-46.70908682	726
Av Afrânio Peixoto, 332	-23.56583221	-46.71092488	727
Educação Física II	-23.56361835	-46.71271652	725
Acesso CPTM II	-23.56089915	-46.71286672	727
Raia Olímpica	-23.55664938	-46.72065653	726
Psicologia II	-23.55446113	-46.72476836	727
Portaria 2	-23.55276336	-46.72812916	722
Terminal de Ônibus	-23.55252364	-46.73184268	729
IPT	-23.55603962	-46.73378997	730
POLI - Eletrotécnica	-23.55797092	-46.73268691	728
FAU II	-23.55934715	-46.72986254	731
Geociências	-23.56065575	-46.72720514	730
Letras	-23.56203625	-46.7243848	738
História e Geografia	-23.56408916	-46.72258236	746
Farmácia e Química	-23.56534424	-46.72486089	762
Biblioteca Farmácia e Química	-23.56648745	-46.72702007	770
Rua do Lago	-23.56754214	-46.72897406	781
Biomédicas	-23.56824527	-46.73189901	790
Portaria 3	-23.56798532	-46.74039232	758
Acesso Vila Indiana	23.56860175	-46.73168376	785
Biomédicas	-23.56824527	-46.73189901	790
IPEN	-23.56605844	-46.73851199	778
COPESP	-23.56389124	-46.74072146	772
Acesso à Rio Pequeno	-23.55986653	-46.74189627	745
PUSP-C II	-23.55983654	-46.73939768	750
Física	-23.55976011	-46.73483719	736
Oceanográfico	-23.56098625	-46.73065892	735
CEPAM	-23.56589426	-46.72538126	763
Butantan	-23.56419733	-46.72220416	745
Casa Japonesa	-23.56285874	-46.71959336	734
Paço das Artes	-23.56368213	-46.71628121	731
Academia de Polícia	-23.56544566	-46.71322715	726
R. Alvarenga, 1585	-23.56781012	-46.71176836	727
Av Vital Brasil, 559	-23.57107494	-46.70960113	731
Terminal Metrô Butantã	-23.57213697	-46.70893125	731

O anúncio das informações aos usuários é feito de acordo com a interface gráfica desenvolvida e explicada na seção 5.2.2. As informações disponíveis na tela são: **'Linha'**, **'Próxima Parada'**, **'Tempo Estimado'** e **'Velocidade'**. A maneira como essas informações são obtidas é explicado a seguir.

Para a **'Linha'**, a informação é obtida a partir da escolha do operador que seleciona a linha de ônibus desejada na interface gráfica mostrada pela Figura 18. Por enquanto, foi utilizado apenas a Linha 8022-10: Cidade Universitária - Metrô Butantã. Após testes bem sucedidos é possível implementar o sistema em outras linhas.

A sequência de paradas já está definida, como mostra a Tabela 3. Sendo assim, para realizar a troca de paradas na interface gráfica basta determinar, a partir das coordenadas obtidas pelo módulo GPS em tempo real, se o ônibus chegou na próxima parada. Para tornar isso possível, decidiu-se estabelecer uma região que cercará o ponto referente à parada e, caso as coordenadas obtidas estejam dentro dessa região, significará que o ônibus chegou ao destino. Consequentemente, a informação **'Próxima Parada'** será atualizada.

Um ônibus padrão da Sptrans possui em torno de 12 metros (SPTRANS, 2007). Considerando que os ônibus nem sempre param exatamente no ponto, decidiu-se estabelecer uma região de segurança com um raio de 15 metros com o centro localizado no ponto de ônibus.

A velocidade instantânea pode ser obtida diretamente da aquisição feita pelo módulo GPS e ela é mostrada na lacuna correspondente à **'Velocidade'**.

Para o **'Tempo Estimado'**, foi desconsiderada a influência do trânsito no cálculo, já que esse parâmetro possui características imprevisíveis e tornaria a complexidade do cálculo muito maior. Com isso, o tempo estimado seria dado por:

$$Tempo\ Estimado\ = \frac{Distância\ entre\ os\ pontos}{Velocidade\ média\ no\ trecho}$$

A distância entre os pontos de ônibus pode ser obtida via *Google Maps* e a velocidade média no trecho seria obtida a partir de uma média aritmética das velocidades instantâneas ao longo do trecho percorrido.

6. Conclusões

O sistema de transporte público por meio do ônibus é certamente necessário para uma cidade grande como São Paulo. Apesar de oferecer um serviço satisfatório aos seus usuários, a falta de informações e a dificuldade em se determinar qual a próxima parada ainda é um problema que poderia ser solucionado. A presença de soluções para este problema já está presente em diversos países e sem dúvida, são alternativas que seriam bem aceitas pelos usuários no Brasil.

Baseado nesse fato, o trabalho visou o projeto e implementação de um sistema responsável por anunciar aos usuários do ônibus as próximas paradas e o tempo de viagem estimado. Para tanto, utilizou-se uma associação de *hardware* e *software* baseado na arquitetura *Internet of Things* com a finalidade de coletar dados via GPS, realizar cálculos da distância e da velocidade, estimar o tempo de viagem e realizar o anúncio aos usuários de maneira visual e sonora. O projeto do sistema foi feito a partir de diagramas UML que possibilitaram determinar a interação do sistema com o usuário, a sequência de eventos e a estrutura do *software.* O processo de obtenção da próxima parada e estimativa do tempo de viagem foi simplificado devido à complexidade em realizar esses cálculos em tempo real.

Para que o projeto seja implementado de fato em um ônibus, seria necessário proteger o *hardware* contra impactos e alimentar um monitor LCD em corrente contínua. Esses foram os principais pontos que dificultaram o teste do sistema em movimento. O projeto foi uma prova de conceito com o intuito de mostrar que é possível implementar um sistema que realize o anúncio das próximas paradas de um ônibus de modo automático a partir da localização geográfica via GPS. A partir dos resultados obtidos e descritos neste relatório, conclui-se que o projeto é viável e certamente facilitaria a vida dos usuários de ônibus presentes nas grandes cidade.

7. Referências Bibliográficas

ADAFRUIT. Adafruit Ultimate GPS | Adafruit Learning System. **Adafruit**. Disponivel em: <https://learn.adafruit.com/adafruit-ultimate-gps>.

ARDUINO. Intel Galileo. **Arduino**, 2014. Disponivel em: <http://www.arduino.cc/en/ArduinoCertified/IntelGalileo>. Acesso em: 20 Abril 2015.

BADAMASI, Y. A. **The Working Principle of an Arduino**. Nigerian Turkish Nile University. Abuja, p. 1-4. 2014.

CALVIS, A. GPS Shield Hookup Guide. **Sparkfun**, 2015. Disponivel em: <https://learn.sparkfun.com/tutorials/gps-shield-hookup-guide>. Acesso em: 14 Abril 2015.

DZIEKAN, K.; LOTTENHOFF, K. Dynamic at-stop real-time information displays for public transport: Effects on customers. **Transportation Research Part A: Policy and Practice**, 21 Novembro 2006. 489-501.

FILIPEFLOP. QUAL ARDUINO COMPRAR? CONHEÇA OS TIPOS DE ARDUINO. **Blog FILIPEFLOP**, 2014. Disponivel em: <http://blog.filipeflop.com/arduino/tipos-de-arduino-qual-comprar.html>. Acesso em: 20 Abril 2015.

IME-USP. SPtrans 8012 e 8022. **IME-USP**. Disponivel em: <https://www.ime.usp.br/images/arquivos/imagens/itinerarios.pdf>.

LABORATÓRIO DE GARAGEM. Tutorial: Controlando o MP3 Player Shield por comandos via Serial. **Laboratório de Garagem**, 2013. Disponivel em: <http://labdegaragem.com/profiles/blogs/tutorial-utilizando-o-mp3-player-shield>. Acesso em: 18 Abril 2015.

MEZA, A. J.; LIZÁRRAGA, J. A.; LA FUENTE, E. Framework for Estimating Travel Time, Distance, Speed and Street Segment Level of Service. **Procedia Technology**, p. 61-67, Julho 2013.

MICROSOFT. WIndows IoT. **Microsoft Developer Resources**, 2016. Disponivel em: <https://developer.microsoft.com/pt-br/windows/iot>. Acesso em: 22 Setembro 2016.

MICROSOFT. Windows compatible hardware development boards. **Hardware Dev Center**, 2016**.** Disponivel em: <https://msdn.microsoft.com/library/windows/hardware/dn914597(v=vs.85).aspx>. Acesso em: 15 Novembro 2016.

MICROVGA. MicroVGA Arduino support. **MicroVGA**, 2012. Disponivel em: <http://microvga.com/arduino>. Acesso em: 19 Abril 2015.

ORSINI, L. Arduino Vs. Raspberry Pi: Which Is The Right DIY Platform For You? **Readwrite**, 2014. Disponivel em: <http://readwrite.com/2014/05/07/arduino-vs-raspberry-pi-projects-diy-platform>. Acesso em: 12 Abril 2015.

POLITIS, I. et al. Evaluation of a bus passenger information system from the users' point of view in the city of Thessaloniki, Greece. **Research in Transportation Economics**, 2010. 249-255.

RAY, P.P. A Survey on Internet of Things Architectures. **Journal of King Saud University - Computer and Information Sciences**, 3 Outubro 2016.

RICHARDSON, M.; WALLACE, S. Primeiros Passos com o Raspberry Pi. In: RICHARDSON, M.; WALLACE, S. **Primeiros Passos com o Raspberry Pi**. 1ª Edição. ed. São Paulo: Novatec Editora Ltda., 2013. Cap. 1, p. 18-21.

ROBOTIC-CONTROLS. Adafruit Ultimate GPS Breakout | Robotic-Controls. **ROBOTIC-CONTROLS**, 26 nov. 2013. Disponivel em: <http://robotic-controls.com/learn/sensors/adafruit-ultimate-gps-breakout>.

SECRETARIA EXECUTIVA DE COMUNICAÇÃO. Prefeitura testa ônibus com internet wi-fi, TV e aviso sonoro de paradas. **Prefeitura de São Paulo**, 2013. Disponivel em: <http://www.capital.sp.gov.br/portal/noticia/273#>. Acesso em: 19 Abril 2015.

SECRETARIA MUNICIPAL DE TRANSPORTES. Passageiros Transportados - 2015. **Prefeitura de São Paulo**, 2015. Disponivel em: <http://www.prefeitura.sp.gov.br/cidade/secretarias/transportes/institucional/sptrans/acesso_a_informacao/index.php?p=188767>. Acesso em: 14 Abril 2015.

SOUSA, D. Arduino ou Raspberry Pi? Saiba qual micro PC é melhor para seu projeto. **Techtudo**, 2015. Disponivel em: <http://www.techtudo.com.br/noticias/noticia/2015/04/arduino-ou-raspberry-pi-saiba-qual-micro-pc-e-melhor-para-seu-projeto.html>. Acesso em: 12 Abril 2015.

SPTRANS. Manual dos Padrões Técnicos de Veículos. **Secretaria de Transportes da Prefeitura de São Paulo**, Junho 2007.

TOWNSEND, K. Adafruit Ultimate GPS on the Raspberry Pi. **Adafruit**, 2013. Disponivel em: <https://learn.adafruit.com/adafruit-ultimate-gps-on-the-raspberry-pi/setting-everything-up>. Acesso em: 14 Abril 2015.

VALLE, C. D. Sistema de ônibus em SP transportou 6 mi a mais em 2013. **Estadão**, 2014. Disponivel em: <http://sao-paulo.estadao.com.br/noticias/geral,sistema-de-onibus-de-sp-transportou-6-mi-a-mais-em-2013-imp-,1119375>. Acesso em: 14 Abril 2015.

VELEDAF, T. Armazenamento de dados em cartão SD com Arduíno. **Repositório da Automação**, 2014. Disponivel em: <https://automacaoifrsrg.wordpress.com/2014/05/02/armazenamento-de-dados-em-cartao-sd-com-arduino/>. Acesso em: 19 Abril 2015.

WIECHERT, M. D. Tutorial: Usando a EEPROM do Arduino para armazenar dados de forma permanente. **Laboratório de Garagem**, 2012. Disponivel em: <http://labdegaragem.com/profiles/blogs/tutorial-usando-a-eeprom-do-arduino-para-armazenar-dados-de-forma>. Acesso em: 19 Abril 2015.