Universidade Federal
do Rio de Janeiro
Escola Politécnica

UM SISTEMA DE SENSORIAMENTO REMOTO PARA AUXÍLIO À OPERAÇÃO DO MAGLEV-COBRA

Rodolfo Damiani Albuquerque

Projeto de Graduação apresentado ao Curso de Engenharia Eletrônica e de Computação da Escola Politécnica, Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Engenheiro.

Orientador: Prof. Miguel Elias Mitre Campista, D.Sc.

Rio de Janeiro
Março de 2018

UM SISTEMA DE SENSORIAMENTO REMOTO PARA AUXÍLIO À OPERAÇÃO DO MAGLEV-COBRA

Rodolfo Damiani Albuquerque

PROJETO DE GRADUAÇÃO SUBMETIDO AO CORPO DOCENTE DO CURSO DE ENGENHARIA ELETRÔNICA E DE COMPUTAÇÃO DA ESCOLA POLITÉCNICA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE ENGENHEIRO ELETRÔNICO E DE COMPUTAÇÃO

Autor:

Rodolfo Damiani Albuquerque

Orientador:

Prof. Miguel Elias Mitre Campista, D.Sc.

Examinador:

Prof. Luís Henrique Maciel Kosmalski Costa, Dr.

Examinador:

Pedro Henrique Cruz Caminha, Eng.

Rio de Janeiro

Março de 2018

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
Escola Politécnica - Departamento de Eletrônica e de Computação
Centro de Tecnologia, bloco H, sala H-217, Cidade Universitária
Rio de Janeiro - RJ CEP 21949-900

DEDICATÓRIA

Aos meus pais Ana Lucia e Nelson, a minha amiga e companheira Isabella Cavalcante e aos meus amigos irmãos de vida que me fortaleceram durante essa jornada e me fizeram acreditar que seria possível chegar até aqui.

AGRADECIMENTO

Agradeço aos meus pais por estarem sempre ao meu lado me apoiando em todos os momentos

Agradeço ao professor Miguel por toda a dedicação e paciência durante a orientação deste trabalho. Agradeço, especialmente, por todos os conhecimentos transmitidos, que me acompanharão por toda a vida, não só profissional, mas também pessoal.

Agradeço a todos os amigos que se disponibilizaram a debater questões e ideias referentes a este projeto final.

Por fim, agradeço imensamente ao seleto grupo de amigos que fiz durante o curso, sem os quais não teria conseguido chegar ao final dessa jornada: Lucas Cavazzani, Luis Fernando Mascarenhas, Luis Fernando Mourão, Helio Machado e Gabriel Cruz.

RESUMO

Atualmente, um dos maiores problemas das grandes cidades é o sistema de transporte público que impõe grandes dificuldades aos seus usuários, principalmente em termos de conforto. À luz desta questão, este projeto visa contribuir com linhas de pesquisa atuais para o desenvolvimento de uma solução baseada em tecnologias de Sistemas de Transporte Inteligente (*Intelligent Transportation Systems - ITS*). Tais tecnologias podem se servir de conceitos atualmente comuns na Internet das Coisas (*Internet of Things - IoT*), aplicado ao trem de levitação magnética Maglev-Cobra, desenvolvido na Universidade Federal do Rio de Janeiro. Portanto, o objetivo deste trabalho é aprimorar a instrumentação e permitir o monitoramento remoto do Maglev-Cobra. Para alcançar tal objetivo este trabalho propõe um sistema que deve executar três tarefas fundamentais: coleta de dados através de sensores de baixo custo, transmissão dos dados coletados através de protocolos padrões de rede para armazenamento e exibição dos dados sensoriados em uma interface gráfica amigável. O sistema proposto foi implementado utilizando os sensores DHT22 para medir temperatura, o BH1750FVI para medir a luminosidade e o MPU-6050 para medir aceleração. O experimento em campo foi realizado conectando o trem a um servidor através de uma rede IEEE 802.11, de forma a transmitir os dados coletados pelos sensores via TCP. A implementação do sistema em campo demonstrou bons resultados, sendo possível transmitir os dados do trem ao servidor sem perda de pacote, assim como apresentá-los em uma interface gráfica amigável.

Palavras-Chave: Internet das Coisas, Sistema de Transporte Inteligente, monitoramento remoto.

ABSTRACT

One of the major problems in big cities, nowadays, is the public transportation system which imposes many difficulties onto users, mainly in terms of comfort. In the light of this issue, this project aims to contribute with the current state of the art with the development of a solution based on ITS (Intelligent Transportation Systems). This technology can take advantage of common concepts of IoT (Internet of things), applied to the magnetic levitation train Maglev-Cobra, developed at the Universidade Federal do Rio de Janeiro (UFRJ). The goal of this work is then to improve the instrumentation and permit remote monitoring of the Maglev-Cobra. To accomplish that, we propose a system that must execute four main tasks: gather data through low-cost sensors, transmit the gathered data using standard network protocols for storing and processing in a main server, and display of the sensed data in a friendly graphical interface. The proposed system was implemented utilizing the sensors DHT22 to measure temperature, the BH1750FVI to measure light intensity and the MPU-6050 to measure acceleration. The experiment was to connect the train to a server, in order to transmit the collected data from the sensors through TCP. The field implementation was successful in a way that the data could be transmitted to the server without packet loss. Furthermore, the chosen software Elastic Stack accomplished the goal of processing the received data presenting it in a user-friendly interface.

Key-words: Internet of Things, Intelligent Transportation Systems, remote monitoring.

SIGLAS

IoT - *Internet of Things*

ITS - *Intelligent Transportation Systems*

RFID - *Radio Frequency Identification*

WSN - *Wireless Sensor Network*

IEEE - Instituto de Engenheiros Eletricistas e Eletrônicos

TCP - *Transmission Control Protocol*

UDP - *User Datagram Protocol*

JSON - *JavaScript Object Notation*

HTTP - *Hypertext Transfer Protocol*

SOA - *Service Oriented Architecture*

LESFER - Laboratório de Estudos e Simulações de Sistemas Metroferroviários

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Atualmente, um dos maiores problemas das grandes cidades é o sistema de transporte público que impõe grandes dificuldades aos seus usuários, principalmente em termos de conforto. À luz desta questão, este projeto visa contribuir com linhas de pesquisa atuais para o desenvolvimento de uma solução baseada em conceitos de Internet das Coisas (Internet of Things – IoT). Estes conceitos podem apoiar tecnologias baseadas em Sistemas de Transporte Inteligente (Intelligent Transportation Systems – ITS) [1, 2], como no caso deste projeto final, que utiliza conceitos de IoT a fim de agregar valor ao projeto do Maglev-Cobra.

O Maglev-Cobra, desenvolvido na UFRJ, foi projetado visando revolucionar o setor de transporte, contribuindo para o desenvolvimento de soluções alternativas de baixo custo e de baixo impacto ecológico para o transporte coletivo urbano. O Maglev é um trem de levitação magnética que, através da supercondutividade se move sem contato com o solo, tendo apenas o atrito com o ar. Essa característica o torna silencioso, de baixo consumo de energia e com baixa emissão de poluentes devido ao seu motor linear de indução. Seu custo de implantação, que chega a um terço do custo do metrô, também é muito atraente, tornando o Maglev-Cobra um meio de transporte ideal para grandes centros urbanos [3].

O termo Internet das Coisas nasceu em 1999 e foi proposto pelo diretor executivo do Auto-IDcentre no MIT (*Massachusetts Institute of Technology*), Kevin Ashton [4, 5]. Desde então, apoiado em aspectos mercadológicos, o paradigma ganhou força com o avanço de aplicações e tecnologias baseadas em redes sem-fio. Em 2008, a Comissão Federal de Comunicação dos Estados Unidos (*Federal*

Communications Commision - FCC) aprovou o uso do espaço branco do espectro, antes ocupado por canais de TV, para novas aplicações e serviços [6]. No mesmo ano foi fundada a IPSO (*Internet Protocol for Smart Objects*) [7], aliança formada por grandes empresas de tecnologia para difundir o uso de dispositivos de rede IP. Durante a mesma década, a indústria de Smartphones e o suporte às arquiteturas de rede 3G e 4G para telefonia celular cresceram exponencialmente. Todos esses fatores acarretaram o surgimento de inúmeros serviços e aplicações baseadas em IoT.

Segundo o Grupo de Soluções para Negócios de Internet Cisco (*Cisco Internet Business Solutions Group - IBSG*), a Internet das Coisas foi concretizada entre 2008 e 2009, quando a quantidade de dispositivos conectados à Internet ultrapassou o número de pessoas no planeta. Desta forma, com o número de dispositivos conectados crescendo exponencialmente, este projeto contribui para uma indústria em plena expansão.

1.1 Proposta e objetivos

Com o protótipo do Maglev-Cobra já em funcionamento, o próximo passo é aprimorá-lo para que se torne o mais autônomo possível. A arquitetura fundamental da Internet das Coisas consiste na aquisição de dados através de sensoriamento, transmissão dos dados coletados até um ou múltiplos pontos de armazenamento, processamento e posterior geração de inteligência. No contexto do Maglev-Cobra, a inteligência resultante poderia ser usada para correção das medinas analisadas, garantindo o funcionamento correto e o conforto dos passageiros. O projeto proposto emprega conceitos de IoT através do uso de sensores para coletar dados de interesse internos ao trem, de forma a fornecer estes dados a um gerenciador de serviços no servidor e, então, apresentá-los em uma interface gráfica. O projeto faz uso de sensores conectados a um Arduino, que por sua vez se comunica com um servidor através de um módulo Wi-Fi. A comunicação é feita através de protocolos padrão da Internet de forma a tornar o sistema o mais reprodutível possível e, assim, possibilitar projetos futuros com o mesmo princípio.

O objetivo deste projeto final é monitorar o Maglev através da coleta de dados de temperatura, umidade, luminosidade e aceleração para auxílio a operação do veículo. Esses dados devem também ser apresentados através de gráficos em uma central de monitoramento. Neste sentido, este trabalho propõe a criação de um sistema que deve executar três tarefas fundamentais: coleta de dados através de sensores de baixo custo, transmissão dos dados coletados através de protocolos padrões de rede para armazenamento e processamento em um servidor central e exibição dos dados sensoriados através de uma interface gráfica amigável.

O sistema é aplicado no veículo de levitação magnética Maglev-Cobra, na UFRJ, que conecta o CT1 ao CT2 [3]. O sistema coleta dentro do trem dados de temperatura, umidade, luminosidade e aceleração em três eixos. Os sensores utilizados são o DHT22 [8], para coleta de dados de umidade e temperatura; o BH1750FVI Lux [9], para coleta de luminosidade; e o MPU-6050 [10], para coleta da aceleração; todos conectados a um Arduino UNO [11]. O Arduino estabelece conexão com a central através de uma rede Wi-Fi, utilizando o módulo ESP8266 ESP-01 [12], onde a comunicação é feita através da pilha de protocolos padrão da Internet. A central tem a camada de gerenciamento de serviço Logstash [13], a indexação feita pelo Elasticsearch [14] e a interface gráfica obtida através do Kibana [15], onde serão exibidos os dados enviados pelo Arduino.

1.2 Metodologia

Este projeto propõe um sistema de monitoramento remoto do Maglev-Cobra. Desta forma, é utilizada uma arquitetura em camadas comum a sistemas IoT onde quatro camadas exercem funções específicas para chegar ao objetivo final. Estas camadas são a de percepção, a de recepção/transmissão, a de serviço e a de aplicação.

A camada de percepção, faz a leitura das grandezas de interesse através de sensores e envia os dados à camada de recepção/transmissão. Na camada de recepção/transmissão, composta por um microcontrolador e um módulo Wi-Fi, o microcontrolador executa um conjunto de instruções de forma periódica, lendo os dados dos sensores, armazenando-os em variáveis e controlando um módulo

Wi-Fi para que seja feito o envio ao servidor. No servidor, a camada de serviço recebe os dados da camada de recepção/transmissão e faz a integração com o sistema de geolocalização do Maglev-Cobra, de forma a disponibilizar os dados à camada de aplicação. Por fim, a camada de aplicação interage com a de serviço exibindo os dados coletados em uma interface única.

Em relação à transmissão dos dados, o sistema apresenta o envio de pacotes sem perdas da camada de recepção/transmissão ao servidor. Em relação à camada de serviço, esta demonstra flexibilidade para integrar sistemas de acordo com a necessidade. Já a camada de aplicação a exibe os dados revelando versatilidade nos modos de visualização.

1.3 Organização do texto

Este trabalho está dividido em cinco capítulos. O Capítulo 2 apresenta o embasamento teórico para este projeto final e trabalhos relacionados. O Capítulo 3 descreve o sistema proposto, sua arquitetura e os componentes utilizados para solucionar o problema abordado. O Capítulo 4 detalha os testes e o experimento em campo realizados, apresentando resultados de rede e evidências do processamento e apresentação dos dados no servidor. Por fim, o capítulo 5 conclui este projeto final e propõe possíveis trabalhos futuros.

Capítulo 2

Internet das Coisas

O paradigma da IoT ganhou muita força desde sua primeira menção no MIT com a evolução de aspectos tecnológicos chave para o aprimoramento e aplicação do conceito. No âmbito da interação com o mundo físico, tecnologias como códigos de barra, códigos QR (*Quick Response*) e sensores em geral serviram como base para o aspecto de aquisição de dados na IoT. A partir daí o RFID (*Radio Frequency Identification*) [16] e as Redes de Sensores Sem-Fio (*Wireless Sensor Network - WSN*) surgiram como novas formas de interagir com o mundo físico adquirindo dados por meio de comunicação por radiofrequência. Com o sucesso desse tipo de rede e com o surgimento das redes IoT, outras tecnologias de comunicação foram desenvolvidas com a finalidade de transmitir estes dados adquiridos a outras fases de processamento da IoT. Algumas das principais tecnologias e protocolos são o IEEE 802.15.4, que serve como base para outras tecnologias de comunicação de baixo consumo de energia [17], como o ZigBee, o 6LoWPAN (*Low-power Wireless Personal Area Network*) – este, uma evolução do LoWPAN projetada para usar o IPv6 – o WirelessHART e o Bluetooth [18, 19, 20]. Outro protocolo relevante é o IEEE 802.11, o Wi-Fi, amplamente utilizado para aplicações IoT devido à sua grande popularização [21, 4].

Com a evolução das tecnologias de rede que suportam a IoT muitas aplicações surgiram. Segundo Miorandi et al. [22] as áreas de aplicação que tendem a se destacar podem ser separadas em seis domínios: monitoramento ambiental; negócios inteligentes e gerenciamento de produtos; cuidados de saúde; segurança/vigilância; casas/prédios inteligentes e cidades inteligentes. Cada um desses

domínios possui uma vasta gama de aplicações. No domínio de monitoramento ambiental, destacam-se as aplicações relacionadas à predição de fenômenos naturais através da percepção de anomalias, como previsão de tsunamis, furacões e atividade vulcânica. No domínio de negócios inteligentes e gerenciamento de produtos, o uso de RFID destaca-se como forma de monitorar e otimizar a logística da rede de fornecimento e entrega de produtos. No domínio de cuidados de saúde cresce o uso de *wearables* que possibilitam o monitoramento de sinais vitais e de outros indicadores, como quantidade de passos e calorias queimadas, de forma a criar base de dados e gerar inteligência para o melhor controle da saúde do usuário. No domínio de segurança/vigilância, é cada vez maior o uso de câmeras como forma de aquisição de dados, de maneira a gerar inteligência para que, de uma central, procedimentos de segurança sejam acionados. No domínio de casas/prédios inteligentes, sensores e dispositivos atuadores têm sido amplamente utilizados como forma de otimizar recursos, impactando seus moradores tanto de forma econômica quanto social. No domínio de cidades inteligentes, um dos principais setores é o de transportes inteligentes [23], para o qual este projeto visa a contribuir através da implementação de um sistema de monitoramento em tempo real de um veículo de levitação magnética, o Maglev-Cobra [3].

2.1 Arquitetura típica de aplicações IoT

Para apoiar o desenvolvimento da IoT foi estabelecida uma arquitetura base para a construção das aplicações [24, 19]. O princípio de desenvolvimento de uma estrutura de IoT, então, se baseia em três fatores principais. O primeiro o de que qualquer objeto deverá ser identificável. O segundo é a comunicação entre os elementos da rede, seja ele físico ou não. E o terceiro fator é a interação entre estes elementos [22]. Partindo deste ponto, serão apresentadas as principais arquiteturas.

De forma geral, a arquitetura da IoT é formada por três camadas principais. A primeira é a camada de percepção, a segunda a de recepção/transmissão e a terceira a de aplicação, como visto na Figura 2.1 [19, 24].

1. A camada de percepção é a responsável pela aquisição de dados. Nesta camada, estão presentes tecnologias como o RFID, o NFC (*Near Field Communication*) e sensores em geral, que interagem com o meio físico e transformam grandezas, de forma a adaptá-las ao meio digital [19].

2. A camada de recepção/transmissão é responsável por receber os dados coletados e processá-los para serem transmitidos às suas aplicações. Nela, ainda são gerenciadas as diferentes formas de dados para que a interface com a camada de aplicação ocorra. Nesta camada estão presentes tecnologias de comunicação baseadas em protocolos padrões como o IEEE 802.15.4 e o Wi-Fi.

3. A camada de aplicação, também chamada de camada de negócio, é a camada na qual a interação com o usuário final, que pode ser tanto uma pessoa quanto outro objeto, é realizada. Esta camada recebe os dados da camada de recepção/transmissão e os utiliza para oferecer algum serviço ou realizar alguma operação. Esta camada pode conter, por exemplo, um serviço de armazenamento dos dados como backup em um banco de dados, ou de análise, através de gráficos e indicadores, para avaliar as informações recebidas e gerar inteligência [19].

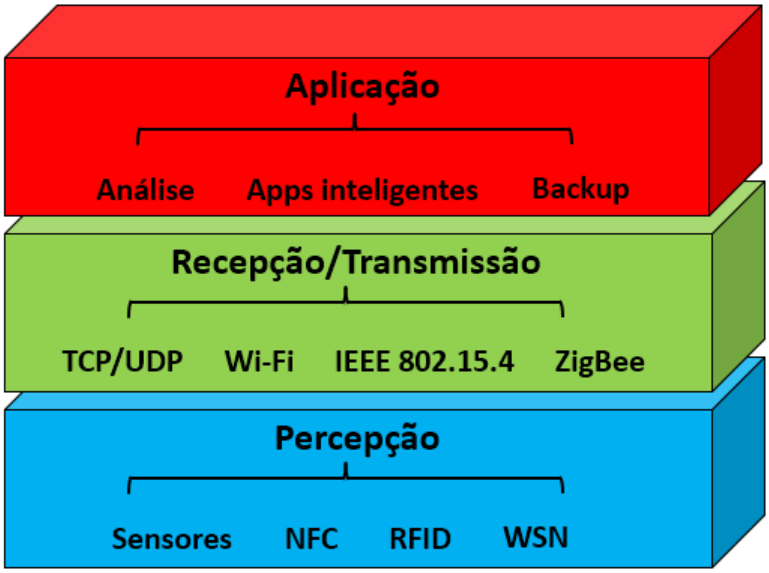


Figura 2.1: Arquitetura em três camadas.

Com a definição da arquitetura em camadas, as aplicações IoT passaram a ser desenvolvidas levando em conta as três camadas existentes. Contudo, com a evolução das aplicações, percebeu-se que seria interessante modificar a arquitetura base para a inclusão de novas funcionalidades nos sistemas IoT. Uma das alterações foi decorrente do problema da heterogeneidade de elementos físicos responsáveis pela aquisição de dados na camada de percepção. As diferenças entre os elementos resultavam em funções e maneiras diferentes de acessar os dados. Dessa forma, uma camada de abstração de objetos tornou-se necessária para que os dados fossem agrupados e padronizados antes de serem transmitidos à próxima camada. Essa função de abstração, então, foi atribuída à camada de recepção/ transmissão [24, 18]. A outra alteração foi decorrente da necessidade de gerir serviços. Uma das principais características de sistemas IoT é que eles devem ser escaláveis. Para isso, deve ser possível adicionar, remover e alterar serviços heterogêneos de forma prática, garantindo-se a preservação da interoperabilidade. Para suprir esta necessidade, foi criada a camada de serviço, baseada em software, entre a camada de aplicação e a de recepção/transmissão. A função da camada de serviço, então, é oferecer abstração de detalhes técnicos das diferentes tenologias presentes no sistema IoT e que não são pertinentes ao desenvolvedor [18], possibilitando um desenvolvimento mais eficiente. Com essas alterações surgiu uma arquitetura orientada a serviço (*Service Oriented Architecture - SOA*), vista na Figura 2.2 , composta de quatro camadas: percepção, abstração de objetos e recepção/transmissão, serviços e, por último, aplicação. Atrelado à camada de serviço, surgiu o conceito de *middleware* aplicado a IoT. Plataformas de *middleware* são projetadas para possibilitar a agregação e filtragem de dados, permitindo que serviços sejam gerenciados de forma conveniente. Além disso, estas plataformas têm a capacidade de tomar decisões, completar informações e entregar serviços utilizando protocolos padrões [24, 18, 20, 19].

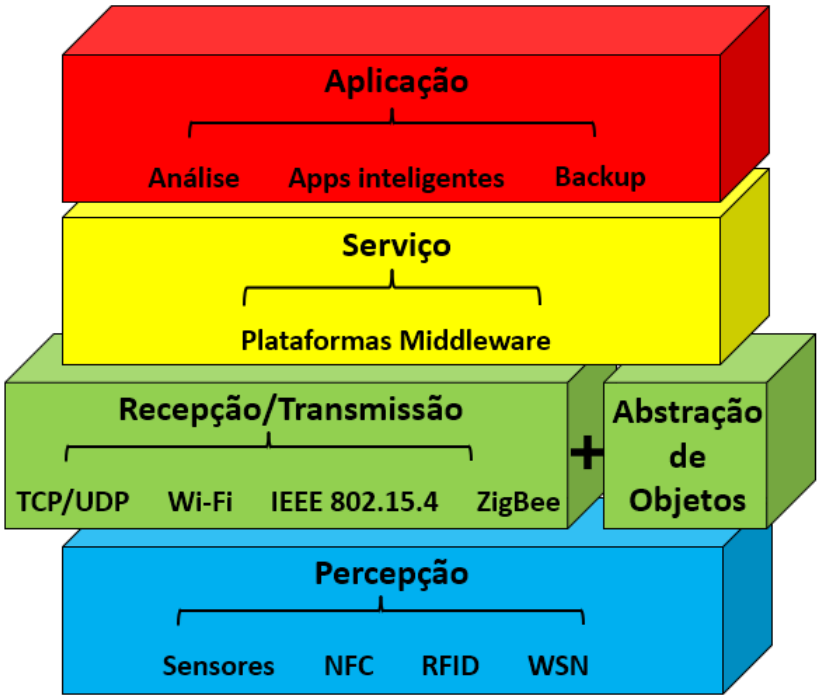


Figura 2.2: Arquitetura em quatro camadas.

2.2 Trabalhos relacionados

Este projeto, tendo como base os conceitos de IoT aplicados ao Maglev Cobra, aborda a questão do sensoriamento remoto do veículo, monitorando em tempo real através de gráficos e indicadores. Com isso, foi feita uma pesquisa sobre trabalhos em IoT, de forma a estudar e entender conceitos chave para este projeto final. Estes trabalhos foram o do Guerrero-Ibanez et al., sobre desafios de integração entre ITS, computação em nuvem e IoT [2], o do He et al., sobre serviços em nuvem para veículos conectados em um ambiente IoT [25], o do Jara et al., sobre um serviço global de descobrimento de dispositivos IoT [26], o do Krishnan et al., sobre computação em névoa [27], o do Vandikas e do Tsiatsis, sobre análise de desempenho de diferentes plataformas IoT [28], o da Caione et al., sobre uma plataforma *middleware* utilizada para fazer com que sensores de celular sejam utilizados para detectar a localização do usuário [29], o do Cruz et al., sobre sensoriamento ambiental em cidades inteligentes [30], o do Bojan et al., sobre tecnologias de ITS baseadas em IoT [1], o do Bajer et al., sobre um sistema de gerenciamento de serviços para IoT [31], e o do Joseph et al., sobre uma plataforma *middleware* para cidades inteligentes [32].

Dentre os trabalhos citados acima, quatro foram selecionados por apresentarem tecnologias, arquitetura e finalidades semelhantes às que serão apresentadas neste projeto final. Dois deles apresentam resultados de projetos voltados a sistemas de transporte e possuem arquitetura semelhante à que será proposta neste projeto final [30, 1], enquanto os outros dois propõem a utilização de *middleware* como solução à questão da interoperabilidade entre componentes heterogêneos. Destes últimos, um apresenta uma proposta voltada à IoT como um todo e o outro tem o foco em cidades inteligentes [31, 32]. A seguir os quatro trabalhos citados são apresentados em maiores detalhes.

O primeiro trabalho citado, o SensingBus, do Cruz et al. [30], propõe uma solução para o sensoriamento ambiental no âmbito de cidades inteligentes. Trata-se de um sistema onde os ônibus da cidade são equipados com sensores com o objetivo de ampliar a área monitorada através da mobilidade. Este projeto conta com uma arquitetura em três camadas: coleta, recepção, e publicação. A primeira é composta pelos nós de sensoriamento embarcados nos veículos. A segunda é implementada através de nós fixos espalhados pela cidade que recebem os dados e os pré-processa. A terceira, situada na nuvem, é responsável pela maior parte do processamento do sistema e pela disponibilização dos dados para os usuários. O segundo trabalho, do Bojan et al. [1], propõe uma solução de integração completa de transportes inteligentes utilizando IoT. Este artigo, com o objetivo de estabelecer uma arquitetura voltada ao setor de transporte, apresenta um sistema em camadas onde: a primeira, de sensoriamento, utiliza GPS para monitorar a localização dos veículos, NFC para monitorar o ingresso dos usuários nos ônibus e sensores para coletar temperatura e umidade dentro dos ônibus; a segunda camada, de monitoramento, utiliza um web-server para receber os dados e um banco MySQL para armazenamento; a terceira, de exibição, conta com um display LCD situado nos pontos de ônibus para que os usuários consigam acesso às informações. Com isso, estes dois trabalhos possuem uma arquitetura em camadas, similar à descrita neste capítulo, aplicada ao setor de transporte, possibilitando um maior entendimento sobre a relação entre IoT e ITS.

Neste projeto final, o Maglev-Cobra, assim como os ônibus no SensingBus, serve como o ponto focal de coleta de dados, com a diferença de que o in-

teresse está no monitoramento interno do trem, enquanto que, no SensingBus, a coleta é direcionada para o monitoramento externo aos carros. Na camada de recepção/transmissão, enquanto que, no SensingBus, são utilizados nós de recepção/transmissão de dados nos pontos de ônibus, neste projeto os dados são transmitidos ao servidor periodicamente ao longo do percurso do trem a partir do nó de sensoriamento. Já no artigo do Bojan et al. [1], diferentemente do artigo do Cruz et al. [30], o sistema proposto tem a coleta de dados direcionada para o interior do veículo, para servir o mesmo propósito do sensoriamento a ser proposto neste projeto de graduação, qual seja monitorar e auxiliar na tomada de decisão para um maior conforto dos passageiros e uso mais eficiente do veículo.

O terceiro trabalho, do Bajer et al. [31], propõe a utilização do Elastic Stack, composto por Logstash, Elasticsearch e Kibana, como forma de processar dados de diferentes fontes, adaptando-os para diferentes serviços, seguindo o conceito de *middleware*. O Logstash é uma plataforma para receber, processar e enviar eventos que podem chegar de diferentes fontes e ser enviados para diferentes destinos. O Elasticsearch, um dos destinos possíveis do Logstash, é um motor de busca que funciona como um banco de dados não sequencial, indexando e armazenando dados em forma de documentos JSON. O Kibana é a interface de visualização dos documentos que são indexados no Elasticsearch. Através do Kibana é possível interagir com os dados indexados, gerando inteligência. O artigo do Bajer et al. [31] explica como o Elastic Stack pode ser usado no contexto de IoT, explicitando o conceito subjacente ao software e detalhes de suas configurações. Neste sistema, proposto por Bajer et al. [31], diversos mecanismos de coleta de dados e outros subsistemas são integrados, dentre eles: sensores, medidores de eletricidade e gás e planta solar. Portanto, para haver a integração destes elementos com o Logstash são utilizados os protocolos UDP (*User Datagram Protocol*), HTTP (*Hypertext Transfer Protocol*) e HTTPS, Modbus TCP (*Transmission Control Protocol*) e Websocket. Do Logstash os dados são enviados ao Elasticsearch de forma a possibilitar a integração com o Kibana e outras aplicações. Este artigo do Bajer et al. propõe um sistema semelhante ao que é usado neste projeto final, utilizando os mesmos softwares. O quarto trabalho, do Joseph et al. [32], também aborda a questão do *middleware*, porém, com o objetivo de estabelecer uma arqui-

tetura voltada para cidades inteligentes. Neste artigo, o *middleware* apresentado é dividido em nove componentes: *gateway*, segurança, roteamento, fila de mensagem, interface de gerenciamento de fluxo de dados, sistema de log e relatórios, painel de controle e barramento de informação. Destes nove componentes, se destacam o *gateway* – sistema de gerenciamento de entradas de dados do sistema – o roteamento e a fila de mensagem – sistemas com a função de encaminhar os dados para seus destinos respectivos – e o barramento de informação, sistema com a função de armazenar os dados e fazer a integração com outros subsistemas. A função destes quatro componentes se relacionam com o que é proposto neste projeto final para receber os dados provenientes do Maglev-Cobra e apresentá-los em uma interface gráfica. Com isso, estes dois artigos possibilitam a maior compreensão da utilização de *middleware* em um contexto IoT, permitindo a integração em alto nível de componentes heterogêneos.

Capítulo 3

Sistema Proposto

O desafio abordado por este projeto é monitorar o Maglev para auxiliar sua operação. Este capítulo apresenta, então, o sistema proposto para realizar tal tarefa. Este capítulo está dividido em duas seções: arquitetura geral e implementação do sistema proposto. Tecnologias relacionadas a IoT e conceitos de ITS são utilizados de forma a apresentar uma solução prática para o problema abordado.

3.1 Arquitetura geral

A arquitetura do sistema proposto, vista na Figura 3.1, tem como base a estrutura em quatro camadas baseada em SOA, onde, além do middleware, também é utilizada a abstração de objetos na camada de recepção/transmissão.

A camada de percepção é responsável por sensoriar temperatura, umidade e nível de luminosidade de dentro do Maglev, assim como sua aceleração nos três eixos. Assume-se que, com esses dados, é possível melhorar o bem-estar dos passageiros e otimizar a operação do veículo ao regular a temperatura e umidade internas, detectar possíveis problemas com a iluminação e ainda verificar se há movimentação brusca no trem. Estes dados sensoriados, então, são enviados à camada de recepção/transmissão.

A camada de recepção/transmissão é responsável pela abstração dos sensores, pela conexão com a rede Wi-Fi e pelo estabelecimento da comunicação com o servidor para a transmissão dos dados sensoriados. A etapa de abstração dos sensores realiza a leitura e o processamento de seus sinais, adaptando-os para a

transmissão. A camada de percepção e a camada de recepção/transmissão ficam localizadas no Maglev e os dados são transmitidos diretamente ao servidor, onde ficam as duas camadas seguintes, de serviço e de aplicação.

A camada de serviço é responsável pelo gerenciamento das entradas, através de uma plataforma de *middleware*, processando-as e as tornando disponíveis para a camada de aplicação, além de registrá-las em um arquivo texto. As entradas do sistema proposto são os dados sensoriados, recebidos através de um socket TCP, e as coordenadas geográficas do Maglev-Cobra, recebidas através de um arquivo JSON. Este arquivo JSON é atualizado com as coordenadas do Maglev-Cobra disponíveis no servidor, utilizadas para rastrear o movimento do veículo. Então, a integração destas coordenadas com o sistema proposto neste projeto final é realizada através desta camada de serviço.

A camada de aplicação, responsável pela apresentação dos dados sensoriados em tempo real, possibilita ao usuário interagir com os dados, gerando gráficos, métricas e mapas, sendo possível, ainda, montar painéis com as visualizações geradas. Tais funcionalidades tornam possível a identificação de tendências e padrões, auxiliando na operação do Maglev-Cobra e mantendo um processo de aprimoramento contínuo do veículo.

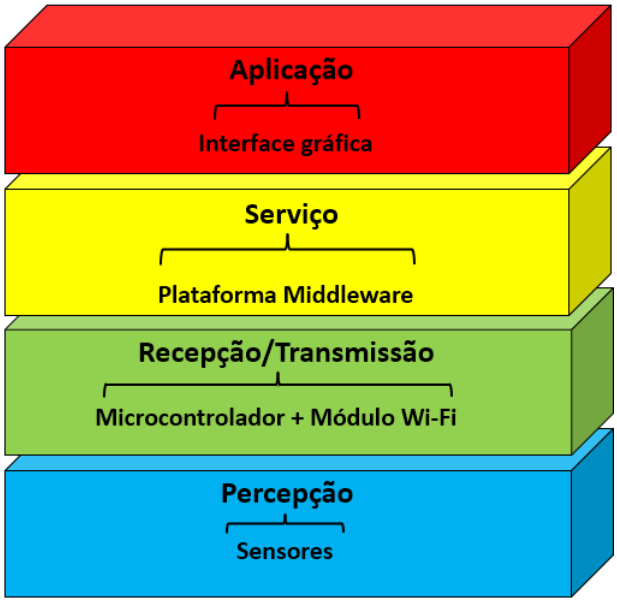


Figura 3.1: Arquitetura do sistema proposto.

3.1.1 Fluxo de dados

Para descrever o sistema, é necessário explicar o fluxo de dados apresentado na Figura 3.2, desde o sensoriamento pela camada de percepção, até a exibição pela camada de aplicação.

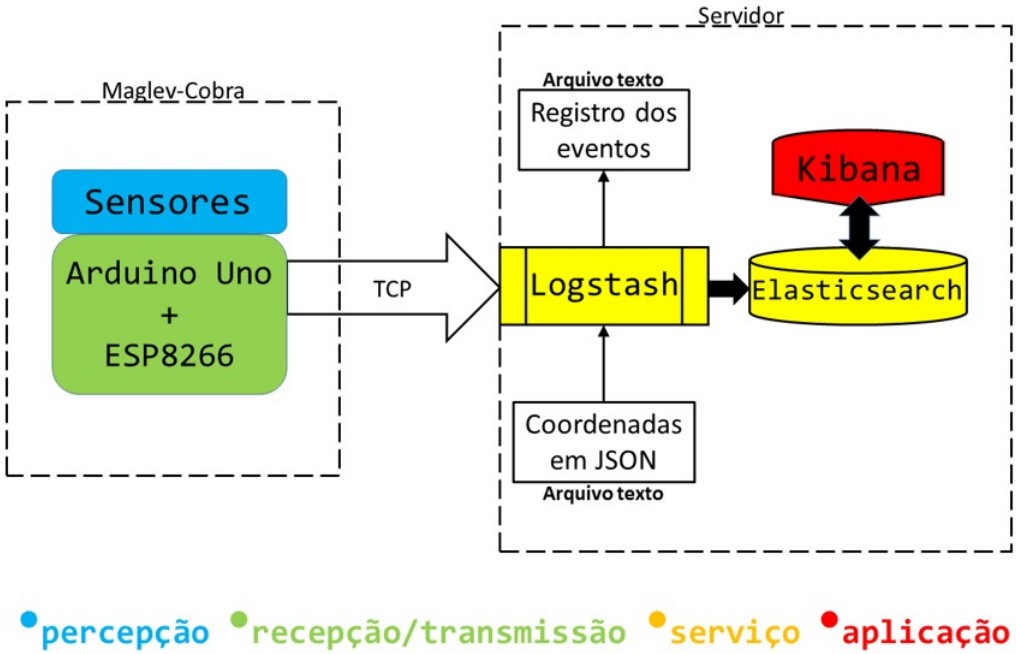


Figura 3.2: Fluxo entre camadas do sistema.

Três sensores fazem a aquisição de dados do sistema: o primeiro coleta temperatura e umidade, o segundo coleta luminosidade e o terceiro coleta aceleração nos três eixos. Estes dados são enviados a um microcontrolador através de barramentos digitais. Ao receber os dados em forma de sinais digitais, o microcontrolador processa-os, armazenando-os em variáveis de acordo com as medidas coletadas. Estas variáveis são utilizadas para compor a string em formato JSON que é enviada ao servidor. A partir daí, o microcontrolador, que também está conectado a um módulo Wi-Fi, emite os comandos para que esse módulo faça a conexão com a rede Wi-Fi e a transmissão dos dados via TCP ao servidor. No lado do servidor, uma plataforma de *middleware* estabelece uma porta para o recebimento de dados da camada de recepção/transmissão via socket TCP e faz a leitura do arquivo de coordenadas sempre que há uma atualização. Dessa forma

gerenciando as duas entradas simultaneamente tratando novos recebimentos de dados como eventos. Após o recebimento de um novo evento, a plataforma de middleware atribui data e hora ao evento e o armazena indexando-o como documento JSON. Por fim, a camada de aplicação consulta os documentos indexados periodicamente apresentando-os em uma interface gráfica.

3.2 Implementação do sistema proposto

O sistema proposto foi implementado com hardware de prateleira e com software gratuito por questões de custo. A seguir, a implementação do sistema é descrita camada por camada.

1. A camada de percepção utilizou o sensor DHT22 para medir temperatura e umidade interna ao Maglev, o BH1750FVI para medir o nível de luminosidade dentro do trem e o MPU-6050 para medir sua aceleração nos três eixos.

2. A camada de recepção/transmissão utilizou o microcontrolador Arduino UNO para realizar a leitura das medições dos sensores, armazenando-as em variáveis, e para o controle do módulo Wi-Fi, e o módulo Wi-Fi ESP8266 ESP-01 para a conexão com a rede e o estabelecimento da comunicação com o servidor para a transmissão dos dados sensoriados.

3. A camada de serviço utilizou o Logstash para o gerenciamento das entradas em forma de eventos e registro destes eventos em um arquivo texto, e o Elasticsearch para a indexação e armazenamento dos eventos em forma de documentos JSON, fazendo a interface com a camada de aplicação.

4. A camada de aplicação utilizou o Kibana para consultar o Elasticsearch periodicamente e apresentar os dados em uma interface gráfica, sendo atualizada automaticamente com um intervalo de tempo pré-definido.

Tabela 3.1: Resumo do hardware e software utilizados na implementação do sistema proposto.

Camada	Hardware	Software
Percepção	DHT22 (temperatura e umidade) BH1750FVI (luminosidade) MPU-6050 (aceleração)	-
Recepção/Transmissão	Arduino UNO ESP8266 ESP-01	-
Serviço	-	Logstash Elasticsearch
Aplicação	-	Kibana

3.2.1 Sensores

O DHT22 é um sensor capacitivo de temperatura e umidade. Este possui saída digital via barramento único, acurácia de $\pm 5\%$ para a umidade e de $\pm 0,5°C$ para a temperatura, faixa de operação para umidade de 0% a 100% e para temperatura de -40°C a 80°C, o que o torna adequado para este projeto. Além disso, este sensor possui um tempo entre leituras de 2 segundos [8].

O BH1750FVI é um sensor de luz ambiente, integrado e com saída digital através de barramento I^2C (*Inter-Integrated Circuit*) [33]. Este integrado é composto por um fotodiodo, um amplificador operacional conversor de corrente em voltagem, um conversor analógico para digital de 16 bits, uma unidade lógica e um oscilador interno utilizado como clock. Este sensor possui dois modos de operação, o de alta resolução, mais lento, e o de baixa resolução, mais rápido. Para este projeto, é utilizado o modo de alta resolução, 1 lux, pois não há necessidade de uma leitura extremamente rápida, além do fato de que este modo oferece uma proteção maior contra ruído [9].

O MPU-6050 atua neste projeto como acelerômetro para medir aceleração nos três eixos. Este sensor, também digital, conta com um conversor analógico para digital de 16 bits, integrado e com interface I^2C. Possui massas de prova separadas, uma para cada eixo, e sensores capacitivos que detectam os desloca-

mentos das massas de forma diferencial. Os principais recursos oferecidos por este integrado, utilizados neste projeto, são a faixa dinâmica programável, que pode assumir os valores ±2g, ±4g, ±8g e ±16g, e a detecção de orientação com sinal [10].

3.2.2 Arduino

No contexto deste projeto, o Arduino é utilizado como uma camada de abstração dos sensores dos quais recebe os sinais digitais processando-os para a transmissão ao servidor. Além disso, este componente também faz a integração com o módulo ESP8266 ESP-01, responsável pela conexão com o Wi-Fi e transmissão dos dados ao servidor.

O Arduino é uma plataforma para aplicações eletrônicas, de código aberto, com recursos de hardware e software. As placas de Arduino são capazes de ler entradas, tanto analógicas quanto digitais, através de diversos protocolos, de forma a processar as entradas e exercer uma determinada ação a partir do código desenvolvido. O código é escrito na linguagem de programação do Arduino, baseada em Wiring, e implementado utilizando o ambiente de desenvolvimento integrado (*Integrated Development Environment - IDE*) do Arduino, baseada em Processing [34].

O Arduino possui diversos tipos de placas, cada uma com capacidades e especificações diferentes, sendo utilizada neste projeto a Arduino UNO. Esta placa possui 14 pinos digitais de entrada/saída – dos quais 6 podem ser usados como saída analógica e 2 podem ser usados para comunicação serial, pino 0, Rx, e pino 1, Tx – 6 pinos de entrada analógica – dos quais 2, pinos A4 e A5, são para entradas SDA (*Serial Data*) e SCL (*Serial Clock*) de utilização do protocolo I^2C – um cristal de quartz de 16 MHz, conexão USB, entrada para fonte de alimentação, um *header* ICSP (*In-Circuit Serial Programming*) e um botão de reconfiguração [11]. Neste projeto serão usadas uma entrada digital para o barramento de dados do sensor DHT22, as entradas A4 e A5 para comunicação I^2C com os sensores MPU-6050 e BH1750FVI, e as entradas 0 e 1 para comunicação serial com o módulo ESP8266 ESP-01.

Além do hardware, este projeto utiliza a IDE do Arduino para programar

o microcontrolador. Como a plataforma em questão é código aberto, existem muitas bibliotecas, desenvolvidas pela comunidade, que podem ser utilizadas para facilitar a codificação da solução. No código desenvolvido foram utilizadas três dessas bibliotecas.

A primeira, `DHT.h`, é utilizada para auxiliar o processamento dos dados do DHT22. Com ela, é definido um construtor passando o número do pino do barramento de dados e o tipo do sensor que está sendo utilizado – `DHT dht(3, DHT22)`. Desta forma, o procedimento de leitura é inicializado através do método `dht.begin()` e os dados são extraídos utilizando os métodos `dht.readHumidity()` e `dht.readTemperature()`. A segunda biblioteca, `BH1750.h`, é utilizada para auxiliar o processamento dos dados do integrado BH1750FVI. Com ela, é definido um construtor – `BH1750 lightMeter` – sem argumentos e o processamento é inicializado através do método `lightMeter.begin`. A partir daí, a luminosidade é extraída com o método `lightmeter.readLightLevel()`. A terceira biblioteca, `MPU6050.h`, auxilia a leitura dos dados do integrado MPU-6050. Com ela defini-se um construtor, sem argumentos – `MPU6050 accel` – e inicializa-se o chip com o método `accel.initialize()`. A partir daí, as acelerações são extraídas com o método `accel.getAcceleration(&AcX, &AcY, &AcZ)` e armazenadas nas variáveis AcX, AcY e AcZ. Por fim, os valores destas três variáveis são divididos por 16384 para que seja apresentado um valor em relação à aceleração da gravidade.

O Arduino, além de fazer a integração com os sensores, faz a integração com o módulo ESP8266 ESP-01. Esta é realizada através de comunicação serial e utiliza comandos do firmware `AT`(*AI-Thinker*) para interagir com o chip ESP. Com isso, o Arduino é programado para controlar o ESP de forma que este faça as conexões e envie as leituras dos sensores ao servidor. O código desenvolvido neste projeto pode ser visto no arquivo `Maglev_Monitor.ino` presente no link da referência [35].

3.2.3 ESP8266 ESP-01

ESP8266 ESP-01 é um módulo de baixo custo, baixo consumo, faixa de temperatura de operação de -40°C a 125°C capaz de conectar microcontroladores à rede Wi-Fi. Este módulo suporta as redes 802.11 b/g/n, muito usadas atu-

almente, podendo trabalhar nos modos cliente, servidor e híbrido, enviando e recebendo dados [36]. A comunicação com o integrado é feita através de porta serial, UART (*Universal Asynchronous Receiver/Transmitter*), no caso do Arduino, utilizando os pinos Tx e Rx. Atualmente existem diferentes bibliotecas e firmwares para operar o ESP8266 ESP-01. Neste projeto, o firmware de comandos `AT` é utilizado.

Com o firmware de comandos `AT`, nenhuma biblioteca é utilizada no Arduino e, sendo assim, os comandos são programados diretamente. Existe uma quantidade enorme de comandos `AT` para as mais variadas configurações e operações com o módulo. Este projeto utiliza os comandos `AT+RST`, para reconfigurar o módulo ao iniciar o script, `AT+CWMODE=1`, para configurar o módulo como cliente, `AT+CWJAP="SSID","SENHA"`, para se conectar com a rede Wi-Fi, `AT+CIPSTART="TCP" "IP_DESTINO",PORTA`, para estabelecer a comunicação TCP com o servidor e `AT+CIPSEND=DATA_LENGTH`, para enviar a mensagem. Após o envio deste último comando, com o comprimento da mensagem, é retornado um ">"indicando que o servidor está pronto para receber o pacote. Só então, a informação é enviada com o método `Serial.println(STRING_JSON)`. Ao final, o comando `AT+CIPCLOSE` é usado para fechar a conexão com o servidor antes do início do próximo laço de repetição. Os comandos utilizados estão no arquivo `Maglev_Monitor.ino` no link da referência [35].

O ESP8266 ESP-01 é um dos modelos de módulos Wi-Fi que utiliza o chip ESP8266EX, dentre outros com desempenhos diferentes e outras funcionalidades. O módulo usado neste projeto é composto por um chip ESP8266EX, um chip SPI (`Serial Peripheral Interface`), um oscilador de 26 MHz e uma antena. Já em relação aos pinos, este módulo possui, além do Tx, Rx, VCC e GND, os pinos RST, para reinicialização, acionado em nível baixo, CH_PD, para gravação ou atualização de firmware, mantido em nível alto em modo normal de operação, GPIO0 e GPIO2, controlados pelo firmware, e um LED para indicar estados de operação [12].

3.2.4 Elastic Stack

O Elastic Stack, ou ELK, é o conjunto de três ferramentas - **E**lasticsearch, **L**ogstash e **K**ibana – utilizado como plataforma IoT neste projeto, apesar de ser usualmente utilizado para análise de logs. Embora o Elasticsearch e o Logstash poderem trabalhar de forma independente, separadamente, as três ferramentas foram desenvolvidas para serem usadas como uma solução integrada. Uma quarta ferramenta, chamada Beats, foi adicionada ao Elastic Stack posteriormente [31]. Esta, contudo, não é utilizada neste projeto. Assim, esta seção descreve qual a função de cada ferramenta, como funcionam e quais as configurações relevantes para este projeto.

O Logstash é um encaminhador de eventos baseado em plugins com funcionalidades ideais em uma arquitetura IoT. Este consegue receber dados de fontes heterogêneas simultaneamente, transformá-los, agrupá-los e enviá-los para destinos também heterogêneos. Esta definição, relaciona-se diretamente com o conceito de middleware aplicado a IoT, se encaixando nas necessidades deste projeto. Cada evento é processado em três estágios: entrada $\rightarrow$ filtro $\rightarrow$ saída.

No primeiro estágio, os plugins são configurados para receber os dados. Nele é possível, por exemplo, configurar uma porta para o recebimento de pacotes TCP, ou então, um arquivo para leitura, onde cada alteração no arquivo configura um evento. Cada uma dessas fontes é identificada por um plugin. Dentro de cada plugin existem as especificações, como por exemplo, a opção "codec", onde é possível definir como o dado é decodificado ao ser recebido [13]. No estágio chamado de filtro, é onde a maior parte do processamento acontece. Nele é possível remover, adicionar e alterar os valores dos atributos do evento. Além disso, como podem existir diversos tipos de entrada, este estágio possibilita alterações condicionais de eventos, o que torna a manipulação dos dados prática ao desenvolvedor. O estágio de saída é onde os destinos dos dados são definidos, que também podem ser dos mais variados, desde um terminal, para identificar possíveis falhas, até o próprio Elasticsearch. Assim como no estágio de filtragem, aqui, o uso de condicionais se mostra necessário, possibilitando, por exemplo, o envio de um comando para uma ação em um sistema com atuador enquanto transmite uma saída diferente ao Elasticsearch. O código

de configuração do Logstash desenvolvido neste projeto pode ser acessado no arquivo `maglevMonitor.conf` no link da referência [35].

A segunda ferramenta, Elasticsearch, é um motor de busca desenvolvido com base no Apache Lucene, com comportamento semelhante a um banco de dados não sequencial, com código aberto e escalável. O armazenamento no Elasticsearch é feito através da indexação de documentos JSON de acordo com o mapeamento definido. Mapear é o processo de definir como um documento JSON, e seus atributos, serão armazenados e indexados. Através do mapeamento é possível definir, por exemplo, quais atributos são somente texto, quais contêm números, datas ou geolocalizações, o formato de um campo de data etc. A ferramenta em questão possui uma API (*Aplication Programming Interface*) dedicada para configurar mapeamentos. Através desta API, são definidos o tipo e os campos de um índice. Para este projeto é definido o índice "maglev"e mapeado o tipo "monitor"com os campos vistos na Figura 3.3. Quando este procedimento é realizado através do Kibana o comando `PUT` é utilizado, já quando é realizado via linha de comando em sistemas Linux a ferramenta `curl` é utilizada [14, 37].

```
PUT maglev
{
  "mappings": {
    "monitor": {
      "properties": {
        "location": {"type": "geo_point"},
        "gForceX": {"type": "float"},
        "gForceY":{"type": "float"},
        "gForceZ":{"type": "float"},
        "temperature":{"type": "float"},
        "humidity":{"type":"float"},
        "luminosity":{"type": "float"},
        "@timestamp":{"type": "date"}
      }
    }
  }
}
```

Figura 3.3: Mapeamento do Maglev via Kibana.

Outra questão importante quando se trata de uma plataforma para IoT é a escalabilidade. O Elasticsearch suporta dois tipos de escalabilidade, a vertical

e a horizontal. A vertical é o método direto de aumentar a capacidade de um mecanismo de busca, ela é realizada aumentando e/ou melhorando os recursos de hardware como núcleos de CPU, memória RAM etc. A horizontal é o método alternativo, aplicada ao aumentar o número de máquinas em um *cluster*. O Elasticsearch é projetado para dividir os registros entre os nós de um *cluster* automaticamente de forma que todos fiquem disponíveis, e, ao ser efetuada uma busca, os resultados de todas as máquinas são agrupados para se alcançar o resultado consolidado. Além disso, para registrar um novo nó em um *cluster* é necessário apenas instalar o Elasticsearch na máquina que será o novo nó e editar um arquivo de configuração [31, 38].

Por último, a terceira ferramenta do Stack, o Kibana, foi projetado como uma plataforma de visualização dos documentos indexados no Elasticsearch. Através do Kibana, é possível acessar os dados armazenados no Elasticsearch e interagir com eles via web. Isto possibilita a geração de inteligência através da exibição e análise dos dados em forma de gráficos, métricas, indicadores e tabelas. No Kibana, existem quatro seções principais - *Discover, Visualize, Dashboard e Management*[31, 15].

Na seção *Management* do Kibana, Figura 3.4, é feita toda a configuração interna e em tempo de execução do Kibana. Nesta seção, os padrões de índices são configurados, e através deles, o Kibana busca os dados no Elasticsearch. Ao configura-los, é possível estabelecer partes coringas no padrão para que este padrão encontre vários índices, como no índice padrão "logstash-*"(Figura 3.4, onde o asterisco representa a configuração coringa. Além disso, nesta seção, ao configurar um índice, também é especificado se este utiliza um campo de *timestamp*, e qual é o nome do campo correspondente. A *timestamp* é um campo do tipo data, que contém data e hora, utilizado pelo Kibana para agrupar e filtrar dados, necessário para correlacionar evento e tempo [31, 15].

A seção *Discover*, vista na Figura 3.5 com o índice "random_variables" selecionado, permite a exploração de entradas de dados de forma direta. Através dos padrões de índices configurados na seção *Management*, é possível visualizar toda a entrada de documento JSON que pertence àquele índice. Além disso, é possível realizar consultas para procurar conjuntos de documentos específicos utilizando

a sintaxe do Apache Lucene. Com isso, a quantidade de documentos correspondentes àquela pesquisa e as estatísticas relacionadas aos valores dos campos são exibidos. Se o índice que estiver sendo pesquisado possuir um campo de *timestamp*, um histograma é apresentado no topo da tela, conforme visto na Figura 3.5, com a contagem da quantidade de documentos indexados pelo tempo [15].

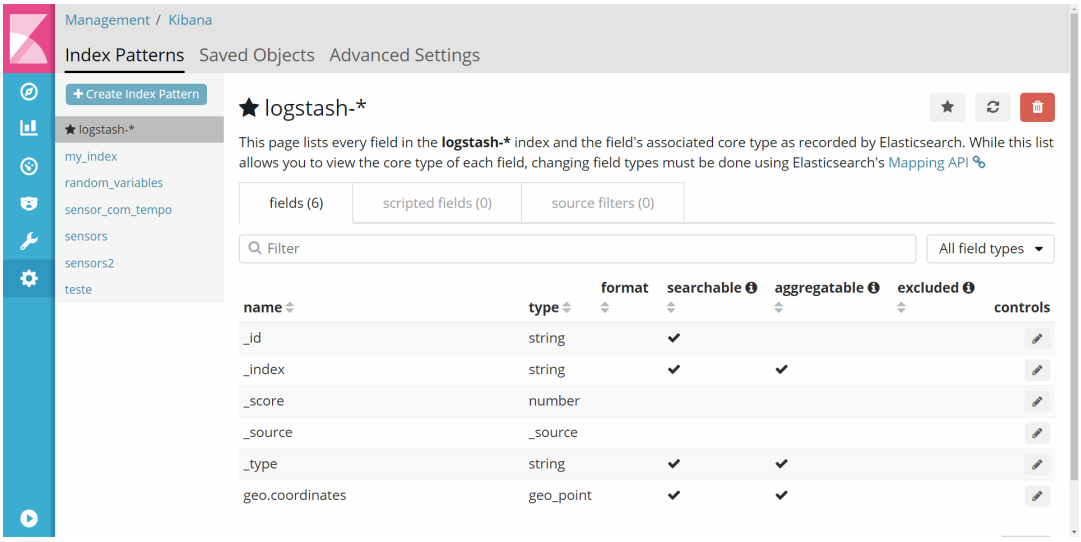


Figura 3.4: Seção *Management*.

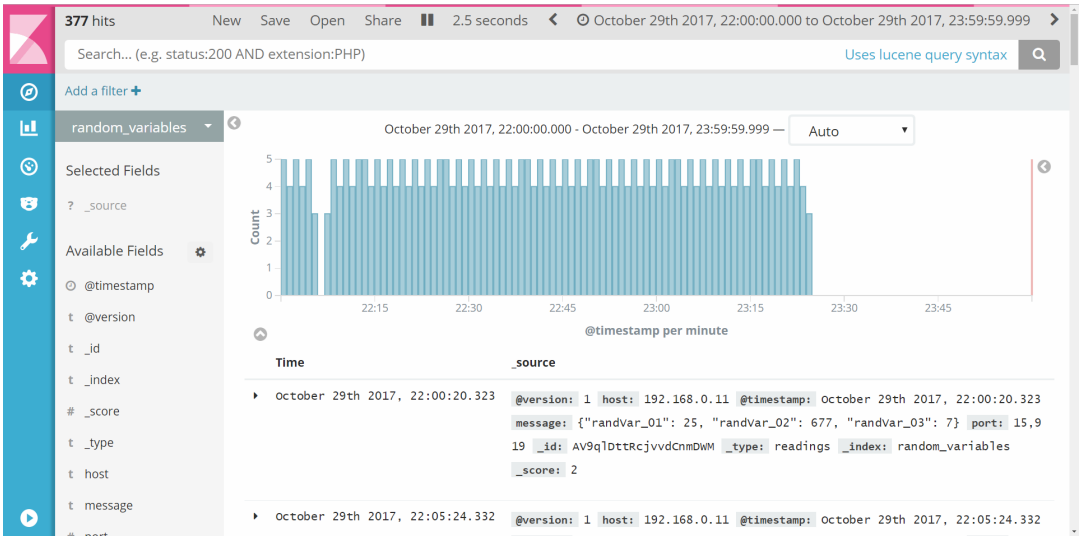


Figura 3.5: Seção *Discover*.

A seção *Visualize* (Figura 3.6) permite que sejam criadas visualizações dos dados que foram indexados no Elasticsearch. Estas visualizações são baseadas em consultas do Elasticsearch onde agregações são utilizadas para extrair e processar os dados relevantes [38]. Dessa maneira, os dados são exibidos em gráficos de diferentes tipos, em mapas, utilizando campos de coordenadas e em métricas com valores absolutos que podem ser calculados de diferentes formas. No gráfico em linha da Figura 3.6 os valores da variável "randVar_01"são apresentados em relação ao tempo. Com as visualizações estabelecidas, é possível, então, juntá-las para montar um painel de controle [31, 15].

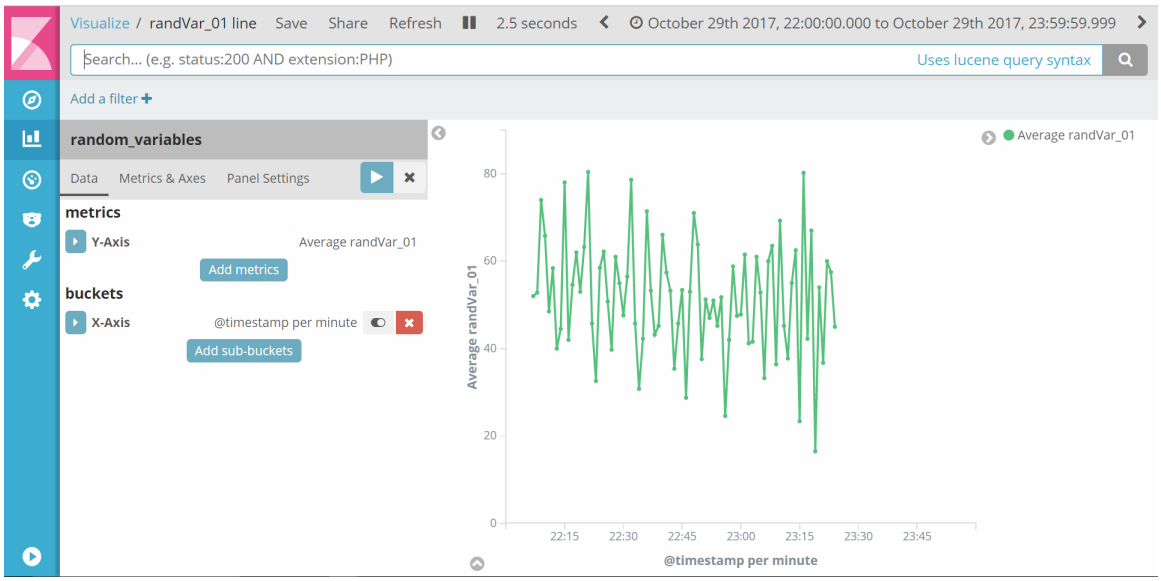


Figura 3.6: Seção *Visualize*.

O painel de controle é montado na seção *Dashboard*, visto na Figura 3.7 com o painel configurado para exibir os gráficos em linha das variáveis "randVar_01"e "randVar_02", assim como seus valores mais recentes em forma de métricas. Nesta seção, é possível agrupar as visualizações que foram salvas para compor um painel onde todos os gráficos, métricas, tabelas, etc. são exibidos em tempo real. O modo edição permite dimensionar e posicionar as visualizações de forma prática e conveniente. Além disso, com um *dashboard* salvo, existe a opção de compartilhá-lo através de um link direto ou integrá-lo a uma outra página web. Nesse caso, tanto o link direto quanto o *iframe* de integração – uma janela a outra página

web – podem ser uma versão atualizada com a última versão salva ou simplesmente um *snapshot* de um momento específico. Estas opções podem ser vistas na Figura 3.8 [15].

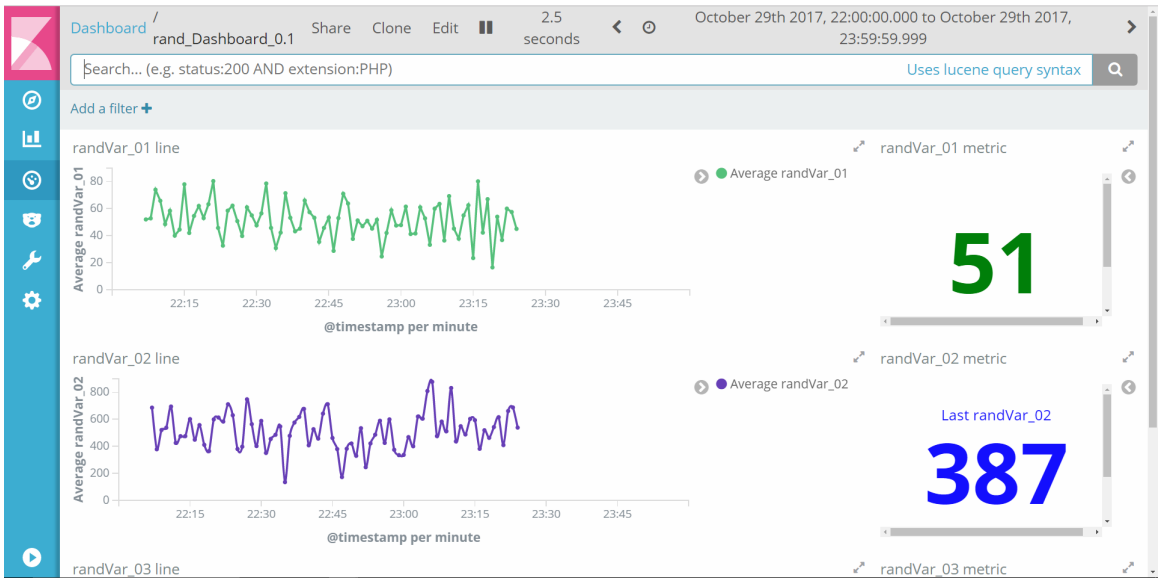


Figura 3.7: Seção *Dashboard*.

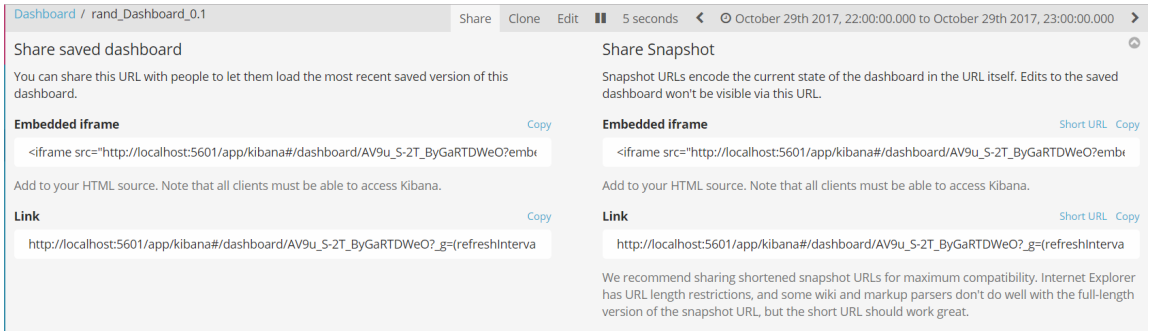


Figura 3.8: Opção de compartilhar o painel.

Além dessas quatro seções principais, o Kibana também possui a aba *Devtools*, vista na Figura 3.9, onde é possível acessar o *console*. O *console* é um plugin que oferece uma interface de interação com a API REST (*Representational State Transfer*) do Elasticsearch. Neste plugin existem duas áreas, o editor e o painel de resposta. O editor, à esquerda na Figura 3.9, permite compor requisições ao Elasticsearch, elas podem utilizar comandos padrões HTTP como PUT, GET e POST.

O uso do comando PUT permite, inclusive, a criação de um índice e um mapeamento no Elasticsearch (Figura 3.3). O painel de resposta, à direita na Figura 3.9, exibe a resposta do Elasticsearch às requisições em formato JSON [15].

Figura 3.9: Seção *Dev Tools*.

Por último, um dos principais recursos do Kibana está no *Time Picker*. Posicionado na barra de ferramentas, no canto superior direito, este recurso permite configurar filtros de tempo e a atualização automática da página. O filtro de tempo restringe os resultados das consultas à uma janela específica de tempo e, portanto, pode ser configurado com as opções *Quick* (Figura 3.10), *Relative* (Figura 3.11) e *Absolute* (Figura 3.12). Por padrão, o filtro é configurado para entradas dos últimos 15 minutos, mas por ele é possível controlar onde a janela de tempo vai estar e qual largura ela terá. O *autorefresh* permite que gráficos de linha, por exemplo, sejam atualizados rapidamente, com intervalos padrões de 5 segundos a até 1 dia (Figura 3.13), sem a necessidade de executar uma nova busca de forma manual [15].

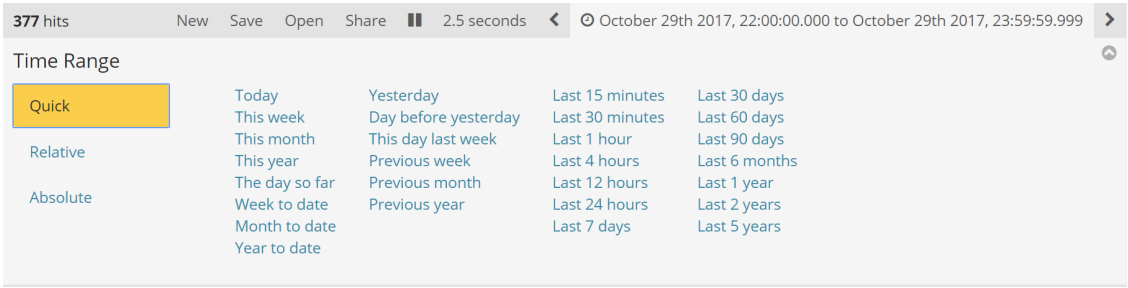


Figura 3.10: *Time Picker* na opção Quick.

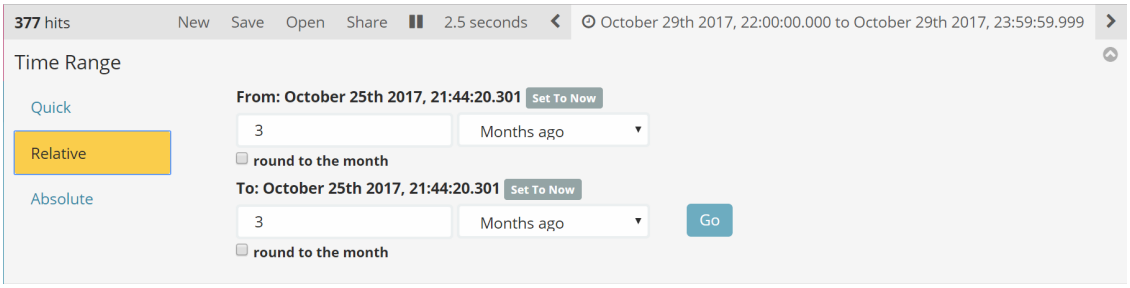


Figura 3.11: *Time Picker* na opção Relative.

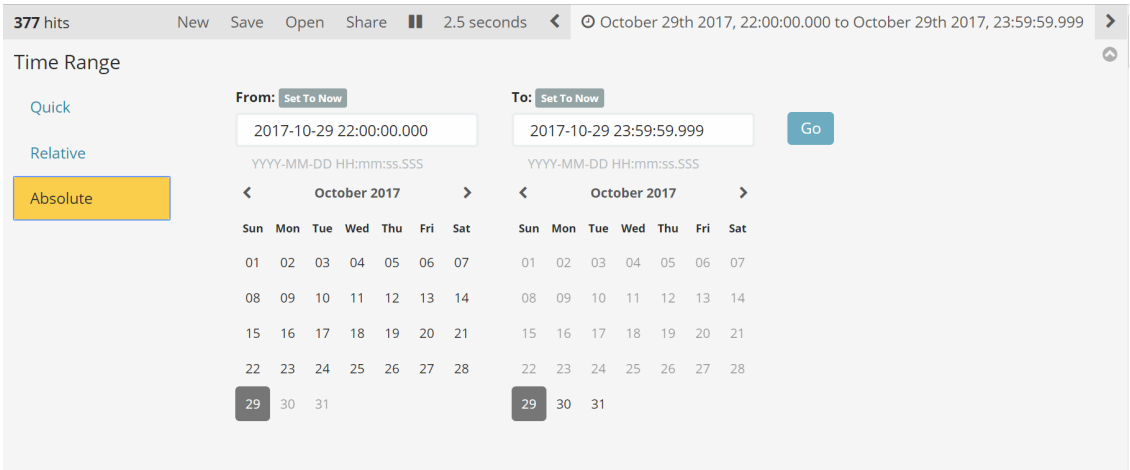


Figura 3.12: *Time Picker* na opção Absolute.

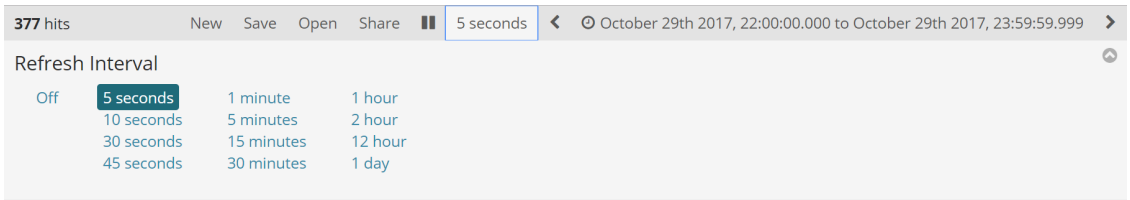


Figura 3.13: Opções de atualização automática.

Capítulo 4

Experimentos

Este capítulo aborda os resultados da implementação deste projeto final. Para isso, os resultados dos testes iniciais e os resultados da experimentação em campo, com o sistema completo, são apresentados. Os ajustes para o funcionamento do sistema em campo e a exibição na interface web também são descritos nas seções a seguir.

4.1 Resultados preliminares

Os testes iniciais foram realizados com o intuito de verificar o funcionamento correto dos componentes, assim como entender como se integram. Esta fase foi dividida em três etapas:

1. Ativação dos sensores conectados à placa Arduino UNO de forma a fazer a coleta e a leitura dos dados através do monitor serial do software do Arduino. Esta etapa foi realizada em um ambiente controlado para que as leituras não fossem comprometidas por ruídos.

2. Integração do Arduino com o módulo ESP8266 ESP-01, de forma a estabelecer conexão com a rede Wi-Fi, e a comunicação com uma máquina local, fazendo o envio de dados randômicos. Nesta etapa foi executado um script em Python no servidor, que configurava um socket para receber os dados e escrevia-os na saída padrão, de forma a verificar o recebimento correto.

3. Instalação do Elastic Stack na máquina local, de forma a configurá-lo para

o recebimento dos dados do Arduino, bem como para a apresentação destes dados na interface gráfica. Esta instalação foi realizada em uma máquina manipulada localmente com o sistema operacional Windows 10. Além disso, o software Postman foi utilizado para para enviar requisições *POST* ao Elastic Stack, testando suas funcionalidades e configurações.

Assim, com as etapas mencionadas acima executadas com sucesso, a integração foi feita de forma a concluir o fluxo de dados, desde a coleta dos sensores, até a apresentação dos dados na interface gráfica. Para este fluxo do cenário de teste inicial, os dados foram enviados a máquina a cada 20 segundos, evitando, com folga, possíveis erros por tempo de leitura dos sensores e pela taxa de envio do módulo Wi-Fi utilizado. Além disso, o tempo de atualização da interface web foi configurado para 5 segundos, garantindo sempre o aproveitamento do evento mais recente. Considerando-se que a implementação deste cenário inicial foi realizada localmente, não se mostrou necessária qualquer alteração na configuração da rede utilizada.

4.2 Resultados experimentais

Nessa seção é descrito o experimento em campo, detalhando os ajustes que foram feitos e os resultados obtidos. Este experimento apresenta algumas diferenças em relação ao anterior, descrito na seção 4.1, onde o sistema é implementado em um ambiente controlado e em uma máquina local, sem a necessidade da interface gráfica estar disponível para a Internet. Em campo, os sensores, o Arduino UNO e o ESP8266 ESP-01 são colocados dentro do Maglev-Cobra de forma a transmitir os dados sensoriados através da rede Wi-Fi MAGLEV para um servidor localizado no laboratório LESFER (Laboratório de Estudos e Simulações de Sistemas Metroferroviários). Assim, o Elastic Stack é instalado neste servidor, com o sistema operacional Debian, de forma a receber os dados e apresentá-los através de uma interface acessível pela Internet.

4.2.1 Configurações

O cenário de experimentação em campo foi realizado de maneira a coletar dados reais do Maglev-Cobra e a apresentá-los em um servidor do GTA (Grupo de Teleinformática e Automação), localizado no LESFER. Para isso, as camadas de percepção e de recepção/transmissão compostas pelos sensores, pelo Arduino UNO e pelo ESP8266 ESP-01 foram colocadas dentro do veículo, de forma a se conectar com a rede Wi-Fi do Maglev-Cobra e a estabelecer comunicação via TCP com o servidor. O Elastic Stack foi instalado no servidor do Maglev-Cobra de forma a receber, processar e exibir os dados enviados de dentro do veículo. Além disso, devido ao fato de que o Maglev-Cobra e seu sistema de GPS (*Global Positioning System*) são acionados apenas uma vez por semana, o que torna o intervalo de tempo para testes reduzido para as circunstâncias deste projeto final, um script em Python foi implementado para simular o movimento do veículo através de sucessivas escritas de 16 coordenadas, pertencentes à linha do Maglev-Cobra, no arquivo texto `coordenadas.txt`, emulando o caso real de envio de dados do GPS para o servidor. A partir daí, foi feita a leitura deste arquivo pelo Logstash e, com isso, realizada a exibição da posição do veículo em um mapa. O script em Python desenvolvido pode ser visto no arquivo `coordinateGenerator.py` através do link da referência [35].

Alguns ajustes tiveram que ser feitos na camada de recepção/transmissão e no servidor para a implementação em campo do sistema proposto.

Na camada de recepção/transmissão, o primeiro ajuste foi na configuração do módulo Wi-Fi para que este se conectasse à rede do Maglev-Cobra e então estabelecesse comunicação com o servidor do GTA para o envio de dados. O segundo foi no tempo entre execuções do script do Arduino, o que antes era perto de 20 segundos, passou para 2,5 segundos aproximadamente, tornando o envio de dados mais frequente para uma melhor análise das medidas no servidor. Dessa forma, para manter o aproveitamento do evento mais recente pela interface, o tempo de atualização automática foi configurado para 2,5 segundos. O tempo entre as execuções do script no Arduino de 2,5 segundos considera os 2 segundos entre leituras do sensor DHT22 mais 0,5 segundo para que todos os comandos sejam processados pelo módulo ESP.

No que se refere ao servidor, o primeiro ajuste foi em relação à instalação do Elastic Stack, pois o sistema operacional utilizado foi o Debian, baseado em Linux, e a versão que estava no servidor não suportava a versão do Java que o Elastic Stack requeria. Por este motivo, foi necessário fazer o upgrade para a versão 8 do Debian e, então, instalar a versão 1.8.0_161 do Java, compatível com o Elastic Stack. O segundo ajuste no servidor foi em relação ao modo de acesso ao Kibana. Como o servidor do GTA no LESFER fica atrás de um *firewall*, foi configurado um *proxy* reverso utilizando o módulo `mod_proxy` do servidor HTTP Apache para tornar o Kibana acessível pela Internet. Para isso, no arquivo de configuração do Apache, foi atribuído um caminho embaixo do diretório `/var/www/` apontando para o endereço do Kibana. Então, no arquivo de configuração do Kibana foi adicionada a configuração `server.basePath` especificando onde montar o Kibana quando este estiver atrás de um *proxy* e tornando a interface acessível através da url (*Uniform Resource Locator*) `http://146.164.26.3/maglev/kibana/`. Outro ajuste, ainda no lado do servidor, foi em relação à implementação do mapa contendo a posição do Maglev-Cobra. Para isto, foi necessária a instalação do X-Pack, um pacote de expansão das funcionalidades do Elastic Stack.

Inicialmente, tentou-se implementar o mapa com a posição do Maglev-Cobra utilizando-se o pacote padrão do Elastic Stack. No entanto, foi verificado que o zoom do mapa deste pacote chega apenas a 10 vezes, o que não é suficiente para visualizar a posição de um veículo com a precisão adequada (Figura 4.1). A partir daí, a solução foi instalar o X-Pack, que possibilita um zoom de 18 vezes, tornando a exibição da posição do Maglev-Cobra adequada para este projeto (Figura 4.2). Além desta característica adicional, o X-Pack também possibilita controle de usuário. Sendo assim, tornam-se necessárias alterações no código do arquivo de configuração do Logstash e do Kibana para que a autenticação com o Elasticsearch seja realizada.

Todos os ajustes na implementação do sistema proposto possibilitaram a coleta de dados reais do Maglev-Cobra, exibindo-os em uma interface gráfica. Isso permite que o Maglev seja monitorado remotamente através de uma interface gráfica única. Os resultados das medidas ao longo do tempo e o painel de exibição dos gráficos são apresentados na subseção 4.2.4.

Figura 4.1: Mapa de coordenadas com zoom 10x.

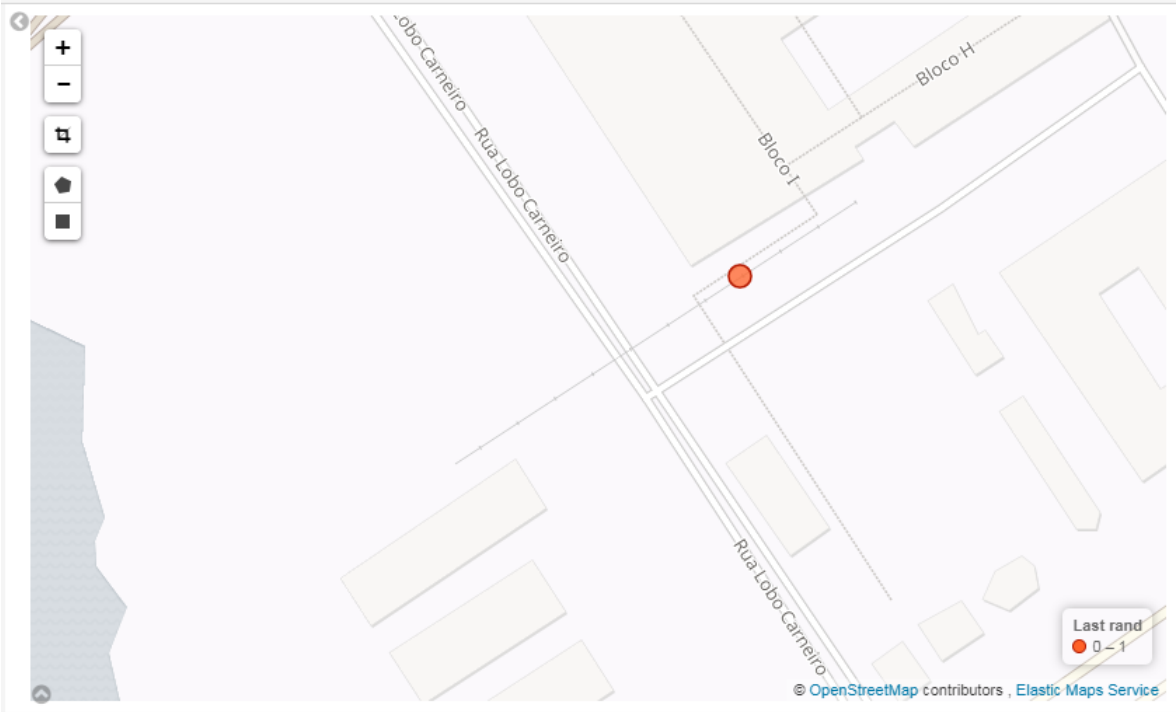


Figura 4.2: Mapa de coordenadas com zoom 18x.

4.2.2 Cenário de experimentação

O cenário de experimentação em campo (Figura 4.3) é formado por três nós: coleta, gateway e servidor. O nó de coleta estabelece conexão Wi-Fi com o gateway, de forma a solicitar um endereço IP. A partir daí, é executado o comando no nó de coleta para estabelecer a comunicação TCP deste nó com o nó do servidor, possibilitando a transmissão dos dados coletados. O nó de coleta, composto pelos componentes da Figura 4.4, é o Maglev-Cobra, o nó de gateway é um roteador TPLINK situado na saída do bloco I para a estação CT1 do Maglev-Cobra (Figura 4.5) e o nó do servidor é uma máquina com sistema operacional Debian localizada no laboratório LESFER.

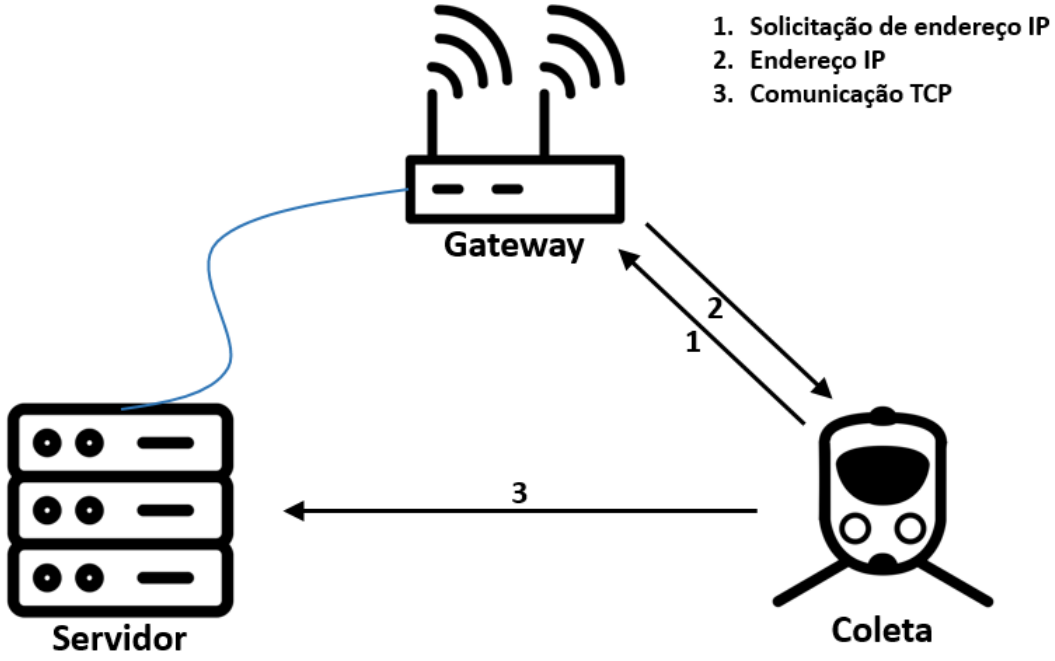


Figura 4.3: Cenário de experimentação em campo.

No circuito responsável pelo nó de coleta (Figura 4.4), os *jumpers* laranja e amarelo conectados aos integrados MPU-6050 e BH1750FVI são responsáveis pela comunicação I^2C com o Arduino UNO. Os *jumpers* azul e branco partindo do ESP8266 ESP-01 fazem a comunicação serial com o Arduino através dos pinos Tx e Rx. O *jumper* branco partindo o DHT22 é responsável pela comunicação digital deste sensor com o Arduino UNO. Além desses *jumpers*, o vermelho e o preto fazem a alimentação da *protoboard*.

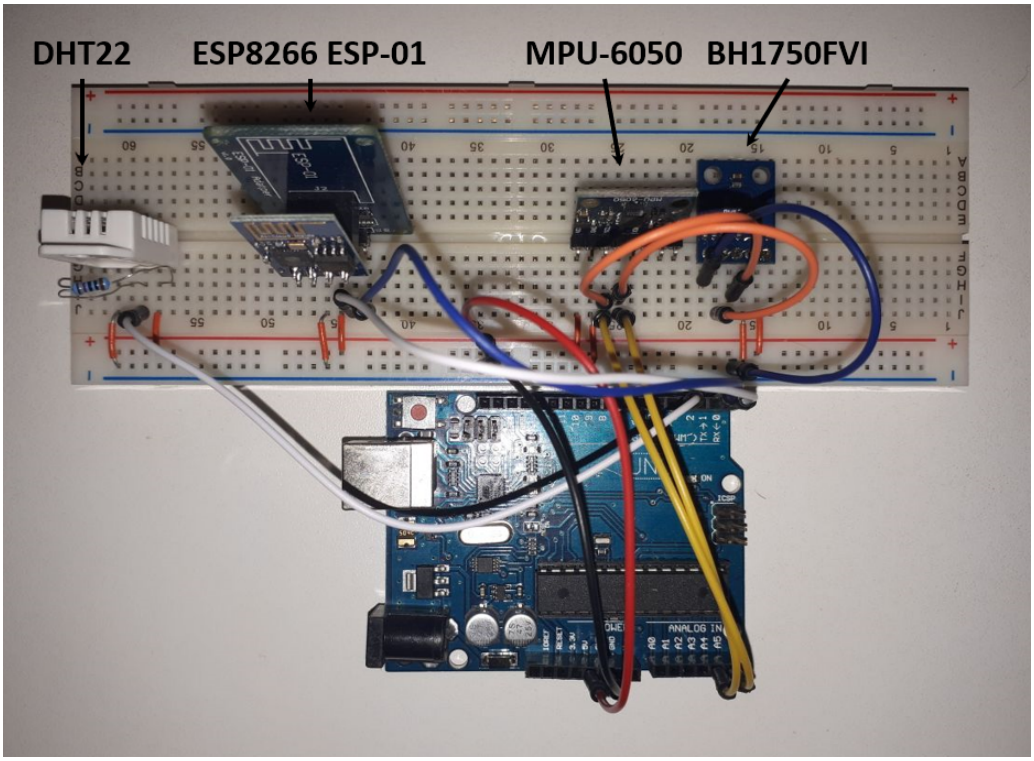


Figura 4.4: Componentes do nó de coleta.

Figura 4.5: Roteador gateway.

4.2.3 Coleta de dados

Procedimentos foram realizados para analisar latência e taxa de transmissão do cenário de experimentação.

A fim de analisar a latência entre o nó de coleta e o nó do servidor do esquema apresentado na Figura 4.3, foi executado o comando `AT+PING` 48 vezes para enviar um sinal de teste partindo do nó de coleta ao servidor. Com isso, foi verificada uma latência média de 6,5 milissegundos com desvio padrão de 2,3 milissegundos.

Para o cálculo da taxa de transmissão foram amostrados 110 pacotes recebidos no servidor em um intervalo de tempo de 5 minutos e 3 segundos. Com cada pacote tendo 1368 bits, a taxa de envio verificada foi de 497 bits por segundo. Para esta análise também foram amostrados os pacotes enviados pelo nó de coleta no mesmo período e não houve perda de pacote.

Para efeito de comparação, os mesmos procedimentos de análise de latência e cálculo de vazão foram realizados no cenário de testes preliminares, descrito na seção 4.1. A latência verificada teve média de 164 milissegundos com desvio padrão de 88 milissegundos. Já a vazão conferida foi de 492 bits por segundo. Também não houve perda de pacote neste cenário.

4.2.4 Resultados das medidas ao longo do tempo

Ao ser acionado o sistema, seu funcionamento é verificado no servidor pelo Logstash através do terminal, onde as entradas são exibidas com o codec *Rubydebug* (Figura 4.6), permitindo verificar os pacotes vindos do socket TCP e as entradas provenientes da leitura do arquivo `coordenadas.txt` no momento em que são processadas. Note que são exibidos os campos retirados do formato JSON junto com os valores atribuídos a eles.

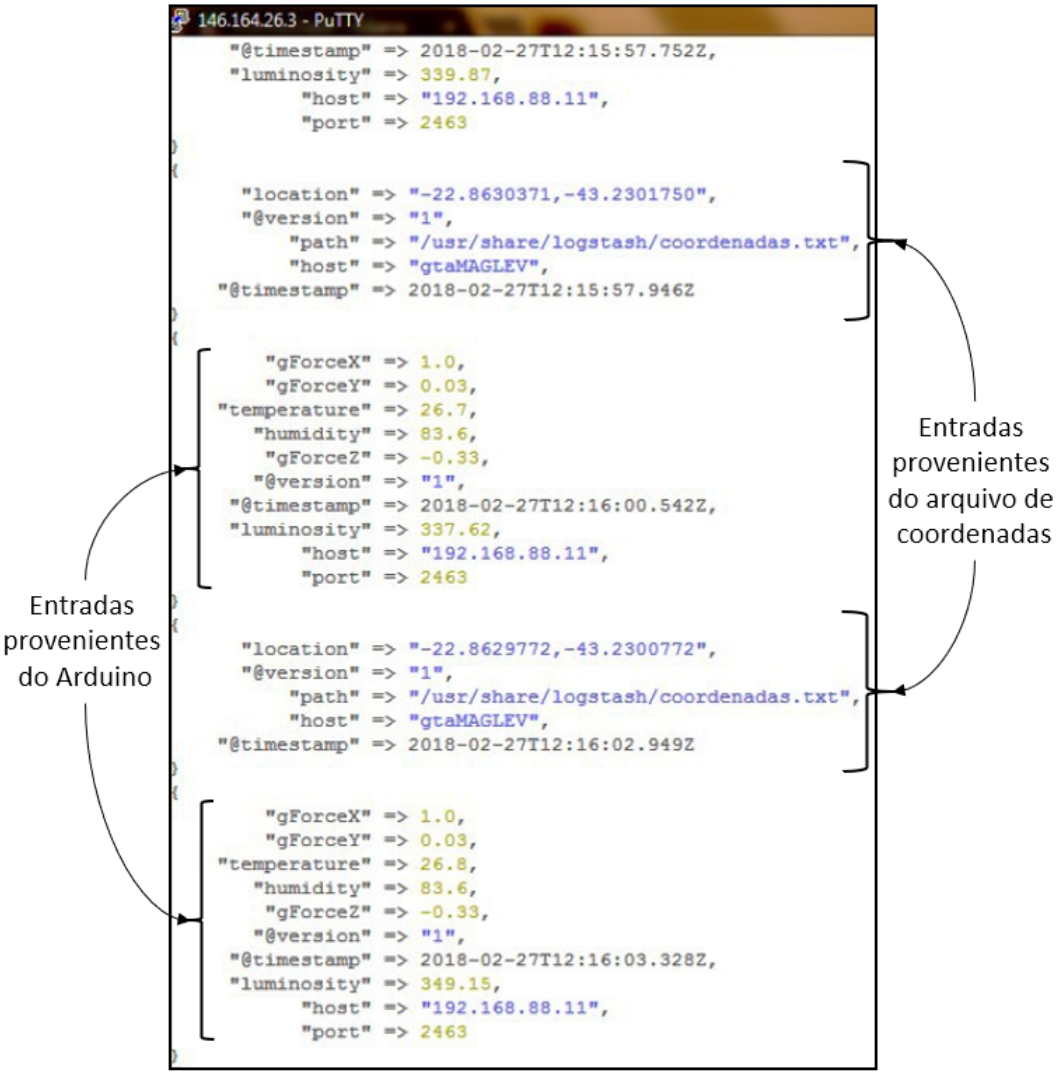
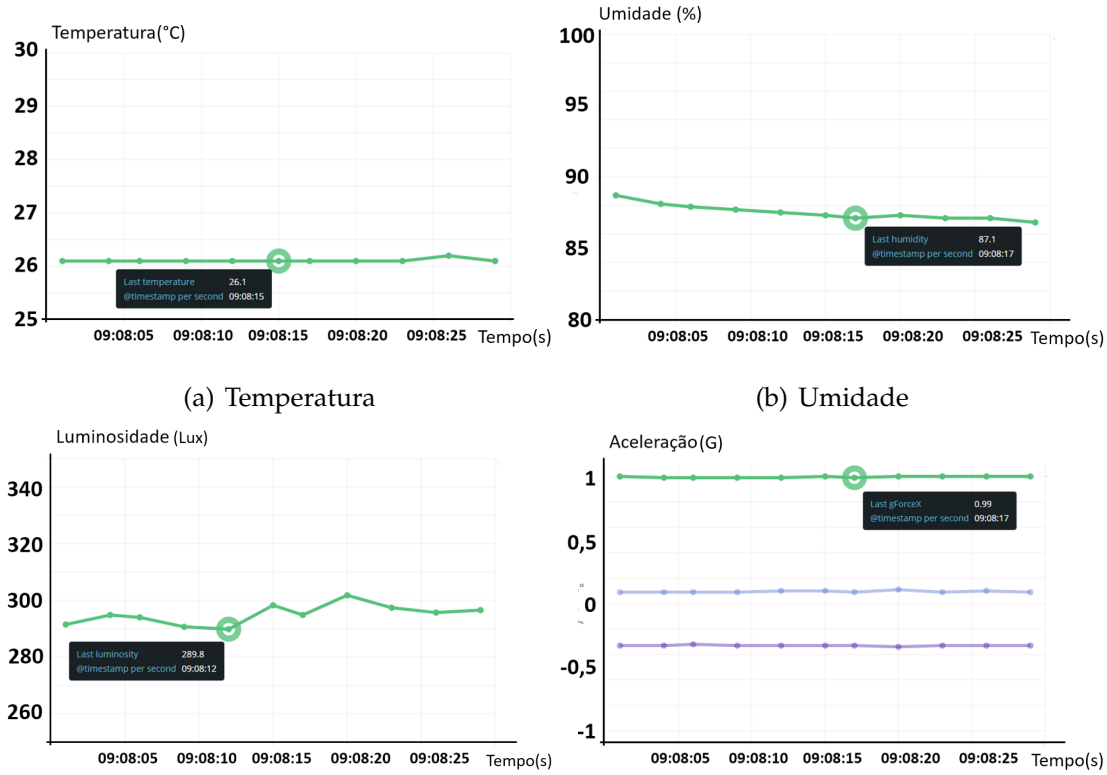


Figura 4.6: Entradas processadas pelo Logstash.

Partindo do Logstash, os dados são encaminhados ao Elasticsearch para que sejam indexados. Com as entradas indexadas torna-se possível a construção de gráficos através do Kibana, de forma a apresentar as medidas dos sensores sobre o tempo (Figura 4.7).



(a) Temperatura

(b) Umidade

(c) Luminosidade

(d) Aceleração

Figura 4.7: Medidas sobre tempo.

Com as visualizações dos gráficos de temperatura, umidade, luminosidade e aceleração salvos, é construído um painel para a exibição dos gráficos. No painel configurado (Figura 4.8), também apresentado no modo tela cheia (Figura 4.9), os valores das leituras mais recentes dos sensores foram adicionados. Além dos gráficos, também foi incluído o mapa com a localização do Maglev-Cobra, conforme apresentado na subseção 4.2.1, Figura 4.2.

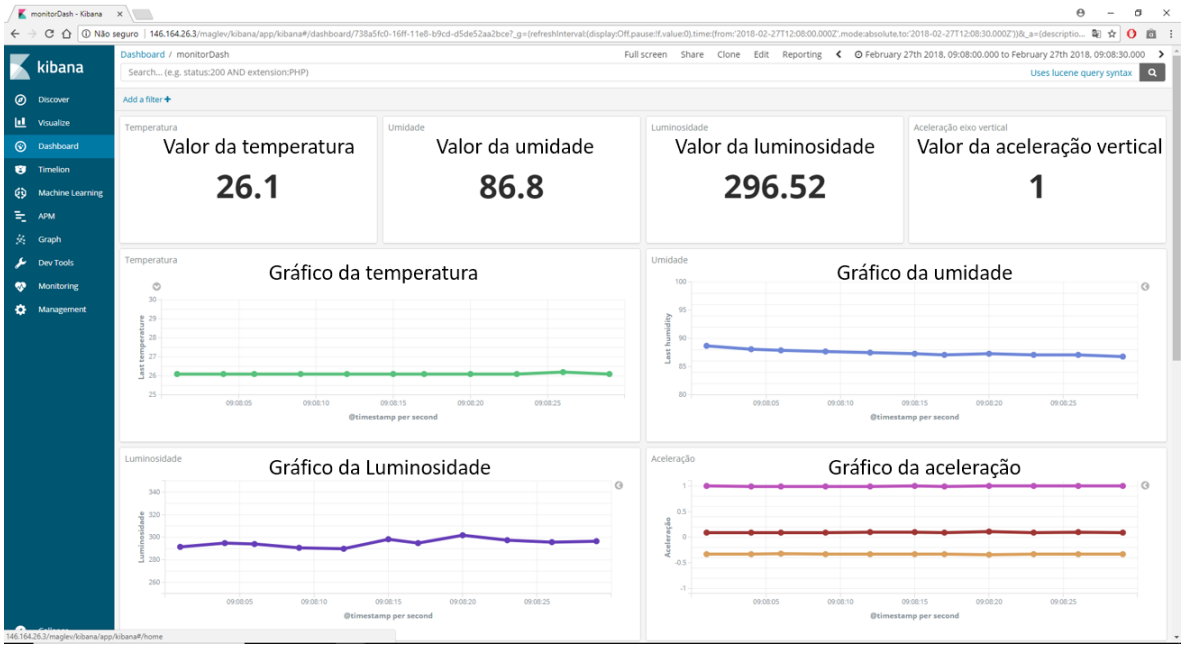


Figura 4.8: Painel com a visualização dos dados dos sensores.

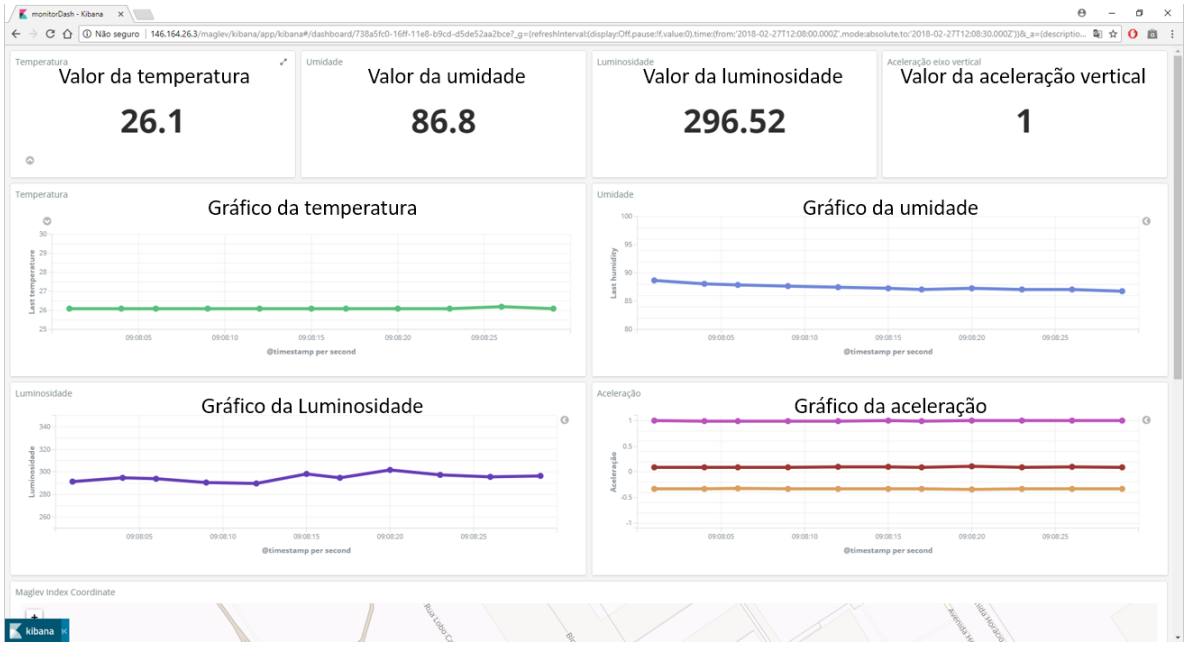


Figura 4.9: Painel em tela cheia.

Capítulo 5

Conclusão

Este projeto abordou a questão do monitoramento remoto do Maglev-Cobra utilizando conceitos de IoT. A solução encontrada para a coleta e conversão das grandezas que seriam analisadas foi a utilização de sensores conectados a um Arduino UNO e a um módulo Wi-Fi ESP8266 ESP-01, de forma a enviar os dados a um servidor. Para o recebimento e exibição dos dados no servidor a solução foi utilizar o Elastic Stack, plataforma normalmente utilizada para leitura e processamento de logs, mas que foi implementada como uma ferramenta voltada à IoT neste projeto.

Durante este projeto, com o embasamento teórico realizado, o Elastic Stack demonstrou ser uma solução que cumpre papéis importantes dentro do conceito de IoT. Através do Logstash foi possível fazer a integração de componentes heterogêneos, permitindo receber dados de diferentes fontes simultaneamente, assim como encaminhá-los para diferentes destinos, garantindo a interoperabilidade entre estes componentes. Através do Elasticsearch foi possível indexar entradas de dados, armazenando-as de forma não sequencial e fazendo a integração com a aplicação Kibana. Através do Kibana, então, foi possível chegar ao objetivo do projeto que é exibir de forma amigável as grandezas coletadas no Maglev.

Um trabalho futuro seguindo no conceito de IoT é a implementação de uma rotina de controle fazendo a correção das grandezas analisadas de acordo com parâmetros previamente definidos. Esse trabalho pode ser realizado com o próprio logstash utilizando estruturas condicionais para enviar comandos à um segundo Arduino ou a um Raspberry caso alguma grandeza ultrapasse um de-

terminado limite. Ainda dentro do conceito de IoT, um segundo trabalho futuro é a ampliação do nó de sensoriamento do Maglev-Cobra, integrando outros tipos de sensores e posicionando estes novos e os que já existem em posições ótimas no veículo para a melhor qualidade de medida.

Referências Bibliográficas

[1] BOJAN, T. M., KUMAR, U. R., BOJAN, V. M., "An internet of things based intelligent transportation system". In: *Vehicular Electronics and Safety (ICVES), 2014 IEEE International Conference on*, pp. 174–179, IEEE, 2014.

[2] GUERRERO-IBANEZ, J. A., ZEADALLY, S., CONTRERAS-CASTILLO, J., "Integration challenges of intelligent transportation systems with connected vehicle, cloud computing, and internet of things technologies", *IEEE Wireless Communications*, v. 22, n. 6, pp. 122–128, 2015.

[3] DE SOUSA, W. T., STEPHAN, R. M., COSTA, F. S., *et al.*, "Projeto MagLev Cobra-Levitaçao Supercondutora para Transporte Urbano", *Revista Brasileira de Ensino de Fısica*, v. 38, n. 4, pp. e4308, 2016.

[4] SURESH, P., DANIEL, J. V., PARTHASARATHY, V., *et al.*, "A state of the art review on the Internet of Things (IoT) history, technology and fields of deployment". In: *Science Engineering and Management Research (ICSEMR), 2014 International Conference on*, pp. 1–8, IEEE, 2014.

[5] EVANS, D., "The internet of things: How the next evolution of the internet is changing everything", *CISCO white paper*, v. 1, n. 2011, pp. 1–11, 2011.

[6] COMMISSION, F. C., OTHERS, "Second Report and Order, FCC 08-260", 2008.

[7] "IPSO Alliance", Disponível em `https://www.ipso-alliance.org/`.

[8] LIU, T., *Digital-output relative humidity & temperature sensor/module DHT22*. Aesong Electronics. Disponível em `https://www.sparkfun.com/datasheets/Sensors/Temperature/DHT22.pdf`.

[9] ROHM SEMICONDUCTOR, *Digital 16bit Serial Output Type Ambient Light Sensor IC*, 2011. Disponível em http://www.mouser.com/ds/2/348/bh1750fvi-e-186247.pdf.

[10] INVENSENSE, *MPU-6000 and MPU-6050 Product Specification*. 1197 Borregas Ave, Sunnyvale, CA 94089 U.S.A, sep 2013. Disponível em https://www.invensense.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf.

[11] ARDUINO, "Arduino UNO R3", Disponível em https://store.arduino.cc/usa/arduino-uno-rev3.

[12] AI-THINKER, *ESP-01 Wi-Fi Module*, 2015. Disponível em https://ecksteinimg.de/Datasheet/Ai-thinker%20ESP-01%20EN.pdf.

[13] ELASTIC, *Logstash Reference*, 2017. Disponível em https://www.elastic.co/guide/en/logstash/6.1/index.html.

[14] ELASTIC, *Elasticsearch Reference*, 2017. Disponível em https://www.elastic.co/guide/en/elasticsearch/reference/current/index.html.

[15] ELASTIC, *Kibana Reference*, 2017. Disponível em https://www.elastic.co/guide/en/kibana/current/index.html.

[16] SARMA, S. E., WEIS, S. A., ENGELS, D. W., "RFID systems and security and privacy implications". In: *International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 454–469, Springer, 2002.

[17] GRANJAL, J., MONTEIRO, E., SILVA, J. S., "Security for the internet of things: a survey of existing protocols and open research issues", *IEEE Communications Surveys & Tutorials*, v. 17, n. 3, pp. 1294–1312, 2015.

[18] ATZORI, L., IERA, A., MORABITO, G., "The internet of things: A survey", *Computer networks*, v. 54, n. 15, pp. 2787–2805, 2010.

[19] LIN, J., YU, W., ZHANG, N., *et al.*, "A survey on internet of things: architecture, enabling technologies, security and privacy, and applications", *IEEE Internet of Things Journal*, , 2017.

[20] DA XU, L., HE, W., LI, S., "Internet of things in industries: A survey", *IEEE Transactions on industrial informatics*, v. 10, n. 4, pp. 2233–2243, 2014.

[21] KEERTIKUMAR, M., SHUBHAM, M., BANAKAR, R., "Evolution of IoT in smart vehicles: An overview". In: *Green Computing and Internet of Things (ICGCIoT), 2015 International Conference on*, pp. 804–809, IEEE, 2015.

[22] MIORANDI, D., SICARI, S., DE PELLEGRINI, F., *et al.*, "Internet of things: Vision, applications and research challenges", *Ad Hoc Networks*, v. 10, n. 7, pp. 1497–1516, 2012.

[23] MEDINA, C. A., PéREZ, M. R., TRUJILLO, L. C., "IoT Paradigm into the Smart City Vision: A Survey". In: *Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), 2017 IEEE International Conference on*, pp. 695–704, IEEE, 2017.

[24] CHOUDHARY, G., JAIN, A., "Internet of Things: A survey on architecture, technologies, protocols and challenges". In: *Recent Advances and Innovations in Engineering (ICRAIE), 2016 International Conference on*, pp. 1–8, IEEE, 2016.

[25] HE, W., YAN, G., DA XU, L., "Developing vehicular data cloud services in the IoT environment", *IEEE Transactions on Industrial Informatics*, v. 10, n. 2, pp. 1587–1595, 2014.

[26] JARA, A. J., LOPEZ, P., FERNANDEZ, D., *et al.*, "Mobile digcovery: A global service discovery for the internet of things". In: *Advanced Information Networking and Applications Workshops (WAINA), 2013 27th International Conference on*, pp. 1325–1330, IEEE, 2013.

[27] KRISHNAN, Y. N., BHAGWAT, C. N., UTPAT, A. P., "Fog computing—Network based cloud computing". In: *Electronics and Communication Systems (ICECS), 2015 2nd International Conference on*, pp. 250–251, IEEE, 2015.

[28] VANDIKAS, K., TSIATSIS, V., "Performance evaluation of an IoT platform". In: *Next Generation Mobile Apps, Services and Technologies (NGMAST), 2014 Eighth International Conference on*, pp. 141–146, IEEE, 2014.

[29] CAIONE, A., FIORE, A., MAINETTI, L., *et al.*, "Exploiting an IoT local middleware for the orchestration of mobile device sensors to detect outdoor and indoor user positioning", pp. 1–5, 2017.

[30] CRUZ, P., DA SILVA, F. F., PACHECO, R. G., *et al.*, "Sensingbus: um sistema de sensoriamento baseado emˆonibus urbanos", *XXXV Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuıdos*, v. 14, n. 1, 2017.

[31] BAJER, M., "Building an IoT Data Hub with Elasticsearch, Logstash and Kibana". In: *Future Internet of Things and Cloud Workshops (FiCloudW), 2017 5th International Conference on*, pp. 63–68, IEEE, 2017.

[32] JOSEPH, T., JENU, R., ASSIS, A. K., *et al.*, "IoT middleware for smart city:(An integrated and centrally managed IoT middleware for smart city)". In: *IEEE Region 10 Symposium (TENSYMP), 2017*, pp. 1–5, IEEE, 2017.

[33] SEMICONDUCTORS, P., "The I2C-bus specification", *Philips Semiconductors*, v. 9397, n. 750, pp. 00954, 2000.

[34] ARDUINO, "What is Arduino?", Disponível em `https://www.arduino.cc/en/Guide/Introduction`.

[35] ALBUQUERQUE, R., "Códigos utilizados neste projeto de graduação", Disponível em `https://github.com/rodolfo-albuquerque/Graduation-Project`.

[36] SEGUNDO, R. T. D. L., MORAIS, R. D. S., "Arduino e módulo ESP8266", , 2018.

[37] TONG, Z., "What is an Elasticsearch Index?", feb 2013, Disponível em `https://www.elastic.co/blog/what-is-an-elasticsearch-index`.

[38] TONG, Z., GORMLEY, C., "Elasticsearch: The definitive Guide", 2015, Disponível em `https://www.elastic.co/guide/en/elasticsearch/guide/2.x/index.html`.