UNIVERSIDADE FEDERAL DE SÃO CARLOS
CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA
DEPARTAMENTO DE COMPUTAÇÃO

LUANA MARIA DA SILVA MENEZES

DESENVOLVIMENTO DE SISTEMA DE MONITORAMENTO IOT PARA RIOS URBANOS DE FÁCIL IMPLANTAÇÃO E BAIXO CUSTO

SÃO CARLOS - SP
2020

LUANA MARIA DA SILVA MENEZES

DESENVOLVIMENTO DE SISTEMA DE MONITORAMENTO IOT PARA RIOS URBANOS DE FÁCIL IMPLANTAÇÃO E BAIXO CUSTO

Trabalho de Conclusão de Curso apresentado ao Departamento de Computação, da Universidade Federal de São Carlos, para obtenção do título de bacharel em Engenharia de Computação.

Orientador: Prof. Dr. Rafael Vidal Aroca

São Carlos - SP
2020

UNIVERSIDADE FEDERAL DE SÃO CARLOS

Centro de Ciências Exatas e de Tecnologia
Departamento de Computação

Folha de aprovação

Assinatura dos membros da comissão examinadora que avaliou e aprovou a Defesa do Trabalho de Conclusão de Curso da candidata Luana Maria da Silva Menezes, realizada em 02/07/2020:

_____

Prof. Dr. Paulo Matias
Universidade Federal de São Carlos

_____

Prof. Dr. Luciano de Oliveira Neris
Universidade Federal de São Carlos

_____

Prof. Dr. Prof. Dr. Rafael Vidal Aroca
Universidade Federal de São Carlos

Dedico este trabalho aos meus familiares que me auxiliam nesta jornada acadêmica tornando-a possível e aos amigos que sempre me motivaram, em especial ao meu grande amigo, Gabriel.

AGRADECIMENTO

Agradeço, primeiramente ao meu Prof. Orientador, o Rafael, pelo suporte durante o desenvolvimento deste trabalho que ocorreu durante todo o difícil período da pandemia de COVID-19, durante os dois meses de semestre suplementar.

Ao pessoal da Prefeitura Municipal de São Carlos, responsável pelo apoio no projeto e instalação, em especial aos colegas da Secretaria de Segurança Pública parceiros deste projeto: Cel. Paulo Belonci, Evandro Mione e Fineias Silva.

“*The whole problem with the world is that fools and fanatics are always so certain of themselves, and wiser people so full of doubts”*

Bertrand Russell

RESUMO

O fenômeno da urbanização, no Brasil, tem desencadeado, – não apenas sozinho, mas em conjunto com outros fatores - na ocupação habitacional de áreas de várzea ribeirinha de forma desordenada. Tais ocupações, porém, oferecem grandes riscos aos seus moradores e transeuntes devido à possibilidade de inundações inesperadas que podem gerar perdas humanas e econômicas. Nesse contexto, diversas ações podem ser tomadas para mitigar esse problema e tem-se buscado na tecnologia novas formas de diminuir os danos gerados por esse tipo de desastre natural intensificado pela atividade humana. Uma delas, que não envolve mudanças estruturais, seria a adoção de sistemas de monitoramento dos rios com sensoriamento remoto, em tempo real, do nível da água e emissão de alertas à população local da ocorrência de possíveis enchentes. Entretanto, atualmente, encontram-se disponíveis no mercado, majoritariamente, soluções proprietárias de alto custo para esse problema e há poucas soluções brasileiras. Destas, destaca-se a ferramenta TerraMA^2, que é de código aberto e desenvolvida por pesquisadores, porém sua manipulação e implantação é complexa, e consequentemente de alto custo devido a necessidade de estudo especializado. Dessa forma, verifica-se a necessidade do desenvolvimento de um sistema que possa atender a essas especificações e que seja de baixo custo, fácil implantação e utilização, visando atender primordialmente a urgente demanda por respostas de redução dos prejuízos causados por enchentes urbanas. Neste trabalho foi iniciada a criação de um protótipo de estação de monitoramento com sensores controlados pelo microcontrolador Arduino. Os dados são enviados em tempo real por MQTT e armazenados em um banco de dados relacional. Foram feitos testes de confiabilidade de medição dos sensores e propôs-se uma solução para os problemas encontrados. Testou-se também a confiabilidade de envio de informações das estações para o servidor na nuvem através de redes móveis. Espera-se que o estabelecimento e outros testes do sistema aconteçam na região central do município de São Carlos (SP), região com importante histórico de alagamentos.

Palavras-chave: monitoramento de enchentes. monitoramento remoto. nível de rios. baixo custo. IOT. MQTT. Arduino. cidades inteligentes.

ABSTRACT

The phenomenon of urbanization in Brazil has triggered - not only alone, but in conjunction with other factors - the housing occupation of riverside lowland areas in a disorderly manner. Such occupations, however, pose great risks to their residents and passersby due to the possibility of unexpected floods that can generate human and economic losses. In this context, several actions can be taken to mitigate this problem and technology has sought new ways of reducing the damage caused by this type of natural disaster intensified by human activity. One of them, which does not involve structural changes, would be the adoption of river monitoring systems with remote sensing, in real time, of the water level and issuing alerts to the local population of the occurrence of possible floods. However, currently, mostly expensive, proprietary solutions for this problem are available on the market, and there are few Brazilian solutions. Of these, the TerraMA2 tool stands out, which is open source and developed by researchers, but its manipulation and implementation is complex, and consequently of high cost due to the need for specialized study. Thus, there is a need to develop a system that can meet these specifications and that is low cost, easy to implement and use, aiming primarily to meet the urgent demand for responses to reduce the damage caused by urban floods. In this work, the creation of a monitoring station prototype with sensors controlled by the Arduino microcontroller was initiated. The data is sent in real time by MQTT and stored in a relational database. Sensors measurements were tested for reliability and solutions were proposed for the problems encountered. The reliability of sending information from stations to the server in the cloud through mobile networks was also tested. The establishment and other tests of the system are expected to take place in the central region of the municipality of São Carlos (SP), a region with an important history of flooding.

Keywords: flood monitoring. remote monitoring. river level. low cost. IOT. MQTT. Arduino. smart cities.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE SIGLAS E ABREVIATURAS

TCC - Trabalho de Conclusão de Curso

IOT – *Internet of Things* ou Internet das Coisas

QoS - *Quality of Service* ou Qualidade de Serviço

SMS - *Short Message Service* ou Serviço de Mensagens Curtas

PWM - *Pulse Width Modulation* ou Modulação de Largura de Pulso

AC-DC - Alternative current/Direct current ou Corrente alternada/Corrente contínua

IDE - *Integrated Development Environment* ou Ambiente de Desenvolvimento Integrado

USB – *Universal Serial Bus* ou Porta Universal

GSM – *Global System for Mobile Communications* ou Sistema Global para Comunicações Móveis

GPRS – *General Packet Radio Services* ou Serviços Gerais de Pacote

UART – *Universal Asynchrounous Receiver/Transmiter* ou Receptor/Transmissor Universal Assíncrono

E/S – Entrada e Saída

SIM – *Subscriber Identity Module ou* módulo de identificação do assinante

Wi-Fi – *Wireless Fidelity* ou fidelidade sem fio

MQTT – *Message Queuing Telemetry Transport* ou Transporte de Filas de Mensagem de Telemetria

GND – *GrouND* ou Terra (eletricidade)

ID – *Identification* ou Identificação

TCP – *Transmission Control Protocol* ou Protocolo de Controle de Transmissão

UDP – *User Datagram Protocol* ou Protocolo de Datagrama de Usuário

FTP – *File Transfer Protocol* ou Protocolo de Transferência de Arquivos

HTTP – *Hypertext Transfer Protocol* ou Protocolo de Transferência de Hipertexto

SUMÁRIO

1. INTRODUÇÃO

A urbanização de algumas regiões próximas a rios, sem planejamento ambiental, muitas vezes, após algum tempo, resulta em regiões que podem sofrer alagamentos ou enchentes. Aliado às mudanças climáticas, o problema se agrava ainda mais. Em São Carlos, especificamente, há um histórico de enchentes que afetam empresas e pessoas. Neste sentido, este trabalho se encaixa no contexto de uma parceria com a secretaria de segurança pública do município de São Carlos, com o intuito de desenvolver um sistema de monitoramento e emissão de alertas de possíveis alagamentos e enchentes em certas regiões de São Carlos-SP.

Parte-se da hipótese de que o monitoramento do nível da água do rio em tempo real poderia ser usado para notificar as pessoas do perigo iminente de acordo com o aumento crescente do nível hídrico e consequentemente do risco de enchente. Além disso, foi prevista a adição de outros tipos de sensores nos pontos de monitoramento do rio para melhor funcionamento e aproveitamento de recursos. Adicionalmente, há uma segunda possibilidade de posterior estudo dos dados coletados do rio para possível entendimento do comportamento do rio e melhoramento do sistema com a construção de técnicas de predição aperfeiçoadas.

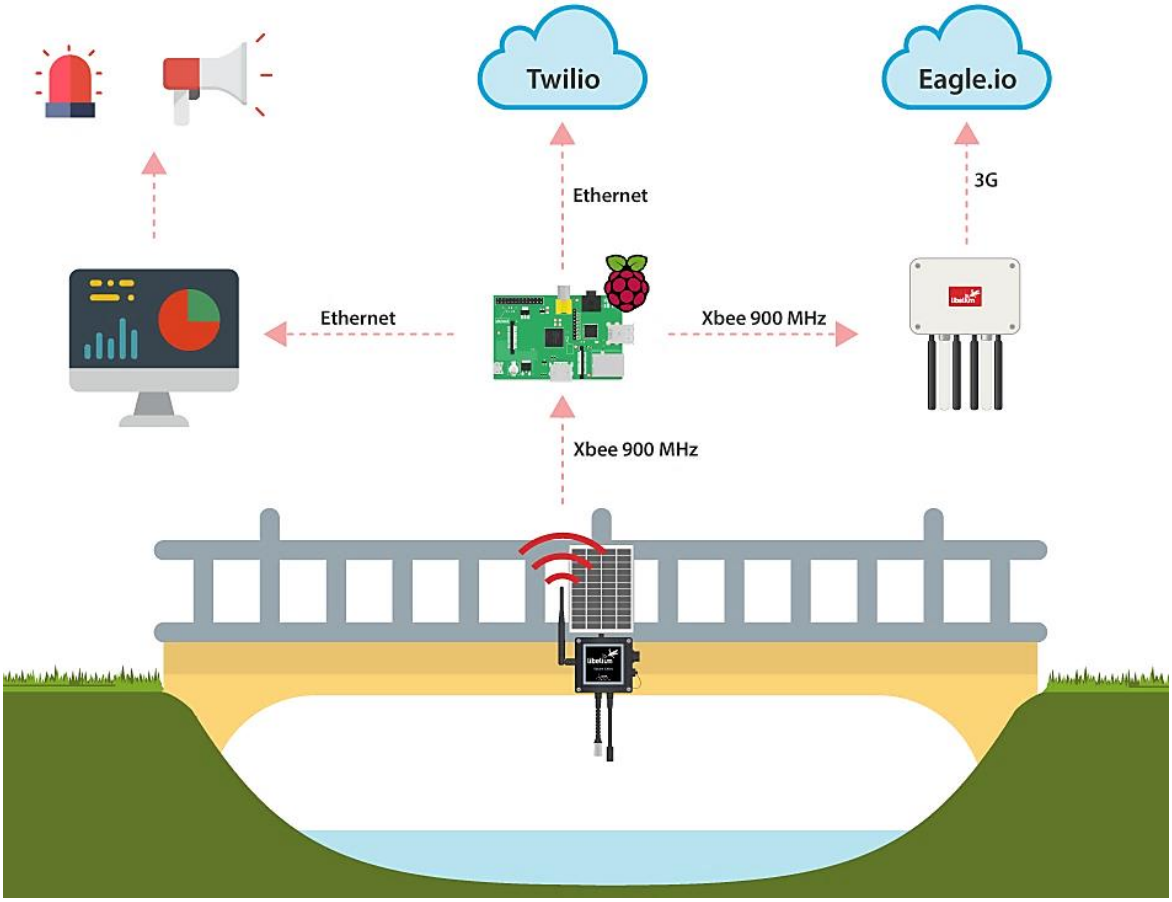
Assim, a justificativa desde projeto decorre da necessidade de prevenção e minimização de danos contra perdas monetárias e humanas em decorrência de alagamentos inesperados através de aviso prévio.

1.1 Objetivo

O objetivo deste trabalho é desenvolver um equipamento (*hardware*) de baixo custo, em parceria com a secretaria de segurança pública da prefeitura municipal de São Carlos, para monitoramento remoto do nível de rios da cidade de São Carlos, coleta e armazenamento de dados históricos e emissão de alertas em situações de possíveis enchentes/alagamentos, para que a população possa agir em tempo. De

forma especifica, o trabalho inclui estudo de tecnologias de microcontroladores, sistemas embarcados e internet das coisas (IOT) aliado à montagem de um circuito com módulos de baixo custo, e a integração com um sistema de recepção de dados em nuvem, usando um servidor padrão MQTT.

Figura 1 - Exemplo de Arquitetura



Fonte: Site Libelium[1]

A Figura 1 mostra uma visão geral do sistema proposto. Um dispositivo instalado nas margens de rio(s) da cidade é monitorado com sensores de nível, e os dados são enviados por telemetria para uma central de monitoramento, em tempo real,

[1] Disponível em: <https://www.libelium.com/wp-content/uploads/2017/12/diagrama_colombia_1100.png>. Acesso em 28 jun. 2020

utilizando, neste projeto, rede de dados da operadora de telefonia celular (GPRS/3G). Os dados são então tratados em um servidor, e podem ser exibidos em displays, telas de alerta, ou serem utilizados para gerar alertas sonoros ou mensagens de alerta para usuários em áreas de risco.

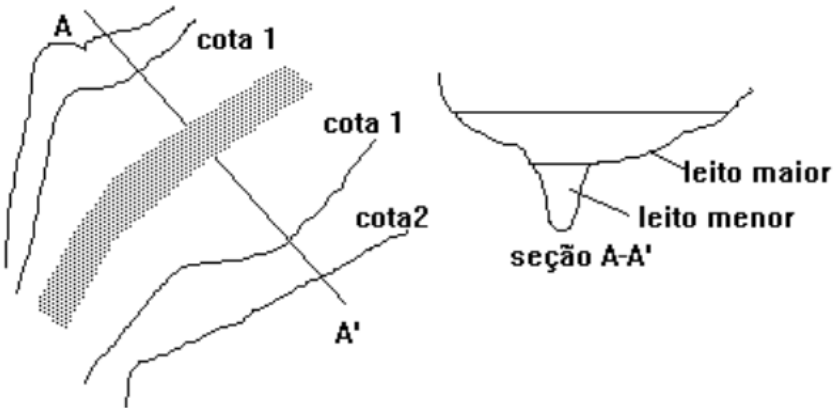
No presente trabalho, foi possível implementar um protótipo completo da solução proposta, com custo aproximado de R$ 431,89, utilizando placa Arduino Mega, módulo 3G SIM800L e monitoramento do nível do rio usando um sensor de ultrassom.

2. FUNDAMENTAÇÃO TEÓRICA

2.1 Contextualização

Enchentes em áreas ribeirinhas são um processo natural no qual o rio inunda sua várzea, principalmente em decorrência de eventos chuvosos, extravasando do seu leito menor para o maior em média a cada dois anos (ver Figura 2). Contudo, com o crescimento populacional das grandes metrópoles esses locais acabam sendo ocupados. Várias são as razões que levam à ocupação desses espaços, duas delas são: ausência de restrição de loteamento de áreas com risco de inundação no Plano Diretor Urbano de praticamente todas as cidades brasileiras e a inexistência de enchentes por longos períodos de tempo faz com que áreas impróprias sejam loteadas por empresários; e a ocupação de áreas de risco pertencentes ao Poder Público pela população mais desprovida de recursos financeiros (TUCCI, 2007).

Figura 2 - Inundação de áreas ribeirinhas



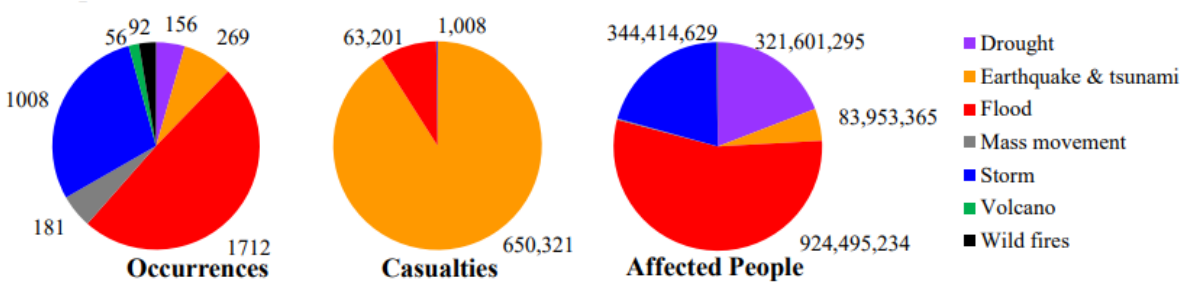
Fonte: TUCCI, 2007.

A situação é agravada ainda mais pelas consequências do desenvolvimento urbano sobre o solo desses locais, como a retirada da vegetação nativa que pode provocar o assoreamento do rio. A impermeabilização da terra, da mesma forma, intensifica o problema. Esta faz com que haja uma diminuição da infiltração de água no solo, impedindo seu escoamento gradual da superfície para o subsolo. Esses fatores acabam aumentando a ocorrência de enchentes.

No mundo todo, milhões de pessoas são afetadas pelas inundações, que já são

reconhecidas como um dos piores tipos de desastre natural. Esse tipo de desastre gera significantes perdas sociais e físicas que podem ter importante impacto na economia de um país (HAPSARI; ZENURIANTO, 2016). Na Figura 3 pode-se notar que as enchentes são o tipo de desastre mais habitual em termos de impacto humano e número de ocorrências.

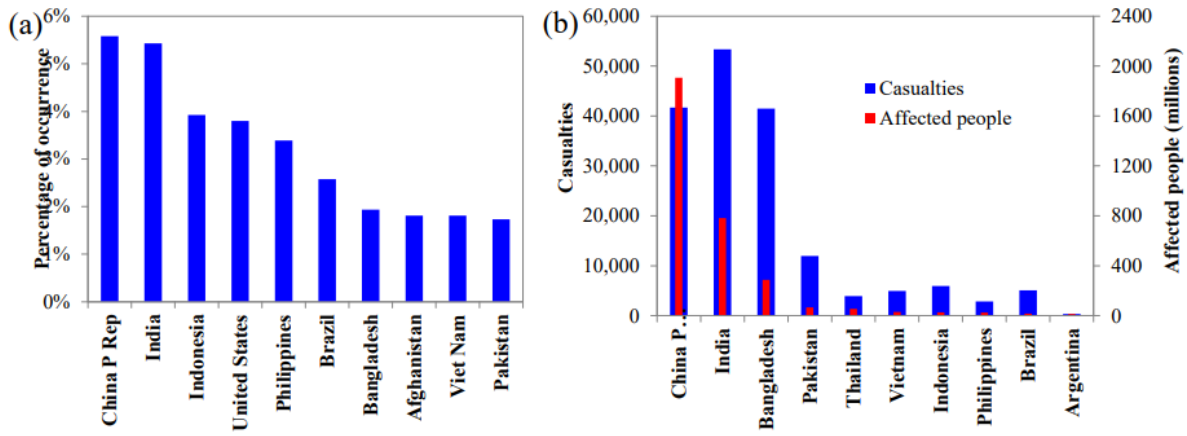
Figura 3 - Desastres naturais em todo mundo (2004-2013)



Fonte: HAPSARI; ZENURIANTO, 2016.

O Brasil ocupa uma posição de vulnerabilidade em relação a enchentes comparado a situação de outros países do mundo. Ele está ranqueado em sexto em termos de números de ocorrências e em nono em número de pessoas afetadas. É possível visualizar essas informações na Figura 4.

Figura 4 - (a) Porcentagem de ocorrências pelo mundo de inundações por país (1970-2011); (b) Número de pessoas afetadas por inundações por país (1970-2011)



Fonte: HAPSARI; ZENURIANTO, 2016.

A cidade de São Carlos (SP) é uma das cidades brasileiras que sofre com essa problemática. Seu principal ponto crítico está no Córrego do Gregório na região

central da cidade. Seu histórico de enchentes remonta do ano de 1905 (segundo registros) até os dias atuais (MAROTTI, 2014). Essa região, mostrada parcialmente na Figura 5, concentra grande parte do comércio do município, o que faz com que além dos danos imateriais, sejam ocasionados grandes prejuízos financeiros. As perdas estimadas em cada uma das inundações mais comuns são de até cerca de R$ 550.000,00 no conjunto de mais de 300 estabelecimentos que ocupam uma área com potencial de enchente por cerca de 5 hectares (MAROTTI, 2014).

Figura 5 – Localização da área de coleta das entrevistas em trecho do Córrego do Gregório em São Carlos/SP. A região corresponde a Av. Comendador Alfredo Maffei, no trecho que se estende entre a Rua Dom Pedro II e a Rua 9 de Julho.

Fonte: MAROTTI, 2014.

O estudo realizado por Marotti (2014) reuniu diversos relatos de pessoas inseridas nesse cenário como comerciantes da região e agente públicos. Lista-se abaixo alguns pareceres acerca dos prejuízos sofridos por este primeiro grupo:

> “Pra mim, mais de 1.000 reais só essa última vez... Essa chuva não foi só aqui que deu, nas loja também! Até carro rodou. Se você ver o tanto que foi até que a gente perdeu pouco.” Jorge, comerciante popular.

> “Prejuízo de bastante mercadoria, geladeira, computadores. Nessa última enchente, né.” Bianca, funcionária de loja do ramo de calçados.

> “Dois funcionários meus tiveram as motos levadas pela enxurrada...” Cristine, gerente de loja de móveis e eletrodomésticos.

> “...a gente tem o prejuízo de que a loja fecha, não pode trabalhar, a gente perde cliente, entendeu? E se, por exemplo, dá de manhã, a gente não pode abrir a loja depois do meio dia com a loja toda suja; a loja tem que estar em ordem pra gente abrir. Perder produto, não, mas... ela fica fechada.” Rita, funcionária de loja do ramo de cosméticos.

> “Já perdi mercadoria. Dessa vez não porque tava tudo no alto” (referente à forte chuva do dia 22 de outubro de 2013) Cíntia, comerciante popular.

Referente ao segundo grupo, o relato do então Diretor de Segurança Pública e Defesa Civil, Pedro Caballero, – ao ser questionado sobre as ações de prevenção que estavam sendo desenvolvidas – mostra-nos que a ideia de criação um sistema de alerta já era cogitada desde a época daquele estudo, em 2014.

> “O que está construído neste momento nós não podemos mexer. A única coisa que podemos fazer de forma imediata é criar um sistema de alerta, porque o risco, o perigo está lá, a vulnerabilidade das pessoas que circulam; essa vulnerabilidade é que temos que eliminar. Se eu tenho o conceito que a ONU me diz que risco é perigo vezes, ou mais a vulnerabilidade, se eu eliminar um desses dois eu já elimino o risco (...). Então a minha ideia e da equipe é trabalhar com sistema de alerta. É preciso ter um bom sistema de hidrometeorologia. Então eu tenho que pegar vários dados; não posso confiar em um só (...) “

Nota-se, entretanto, que os planos por um sistema de monitoramento não foram concretizados, e que o problema ainda persistiu pelos anos seguintes e as inundações continuaram gerando estragos à cidade, como pode-se supor por meio desta nesta notícia de 2018, exposta na Figura 6.

Figura 6 – Manchete em 21/03/2018.

Fonte: Portal de notícias G1[2]

E as enchentes ainda persistem no ano de publicação deste trabalho, em 2020, como pode ser constatado nas notícias presentes nas Figuras 7 e 8. Estas, reforçam o quanto é importante e urgente a realização deste projeto para a cidade de São Carlos.

Figura 7 – Manchete sobre alagamento em 06/02/2020.

Fonte: Portal de notícias G1[3]

[2] Disponível em: <https://g1.globo.com/sp/sao-carlos-regiao/noticia/comerciantes-de-sao-carlos-fazem-liquida-enchente-apos-temporal-invadir-lojas.ghtml>. Acesso em: 17 Sep. 2019.

[3] Disponível em: <https://g1.globo.com/sp/sao-carlos-regiao/noticia/2020/02/06/video-chuva-alaga-e-interdita-a-rotatoria-do-cristo-em-sao-carlos.ghtml>. Acesso em: 6 Maio 2020.

Figura 8 - Manchete sobre enchente em 02/03/2020.

Fonte: Portal de notícias G1[4]

2.2 Trabalhos Relacionados

Esta seção tem como objetivo comentar sobre alguns projetos e pesquisas que buscaram solucionar problemas acerca do monitoramento de rios visando a minimização de danos relacionados a enchentes e inundações originadas por rios.

Dos Reis (2014) apresenta uma solução utilizando um modelo matemático na plataforma brasileira (TerraMA^2) [5] desenvolvida pelo Departamento de Processamento de Imagem (DPI) do Instituto Nacional de Pesquisas Espaciais (INPE). Essa plataforma geotecnológica é usada como base para a construção de sistemas em tempo real de monitoramento e de análise e alerta de fatores ambientais. Para utilizá-la deve-se inserir dados geoambientais, mapas de risco e de vulnerabilidade. O sistema combina esses dados e por meio de modelos de análise as situações de risco são distinguidas e ocorre a emissão automática de alertas de acordo com o nível de risco para seus utilizadores. Devido à complexidade da plataforma na implantação, manutenção e uso, essa alternativa foi desconsiderada devido a demanda por um

[4] Disponível em: <https://g1.globo.com/sp/sao-carlos-regiao/noticia/2020/03/02/comerciantes-deixam-o-centro-de-sao-carlos-apos-enchentes-que-causaram-prejuizos.ghtml>. Acesso em: 6 Maio 2020.

[5] Disponível em: <http://www.terrama2.dpi.inpe.br> Acesso em: 24 jun. 2020

sistema de fácil manejo e baixo custo para, principalmente, rios urbanos que não têm um sistema de monitoramento e não há disponibilidade de grande investimento financeiro pelo Poder Público. Este projeto é focado em fazer um protótipo de monitoramento remoto para rápida implantação em curto prazo. Sistemas mais complexos, como o mencionado, tendem a levar mais tempo para serem implantados e devem ser considerados para projetos de longo prazo.

Intharasombat et al. (2015) propõem um protótipo de baixo custo para detecção instantânea de inundações utilizando um *smartphone* Android, uma placa Arduino, sensores de ultrassom e temperatura comunicando-se através serviço de internet 3G. Um ponto interessante do artigo é a observação e comparação da distância medida pelo sensor de ultrassom HC-SR04 em laboratório e em teste de campo, revelando um importante aumento da taxa de erro nas medições mostrando que bons resultados em laboratório não significam bons resultados na aplicação real caso não haja correta simulação das variáveis presentes no ambiente. Numa primeira investigação acerca dos componentes indispensáveis para este projeto de sistema de monitoramento que prioriza custo e eficiência, a utilização do *smarthphone* aparece aumentar o preço final do protótipo e não ser necessária devido ao uso não previsto de câmeras ou qualquer tipo de análise de imagens neste projeto.

Patil et al. (2019) projeta um modelo de sistema de monitoramento onde há a utilização de um microcontrolador Node MCU ESP8266 Wi-Fi com sensores de ultrassom para medição de nível d'agua, sensor de chuva, de temperatura, de umidade e de pressão. Foi criada uma aplicação Android e um site que exibem - mediante cadastro – informações como status dos sensores, locais seguros e números de pessoas *online*. Não há menção à característica de baixo custo, apesar dos componentes usados assim serem. Verificou-se algumas boas ideias neste trabalho, como a utilização de um sensor de chuva, o site e o aplicativo, entretanto a utilização de Wi-Fi mostrou-se inviável, neste caso, em ambiente real de longa distância, sendo a rede de telefonia celular a mais adequada no contexto deste projeto.

Nos trabalhos analisados acima (excluindo a análise do trabalho já mencionado desconsiderado de Dos Reis (2014)) não há informação sobre a implementação física

dos mesmos. Como mencionado em Intharasombat et al. (2015), sensores de distância podem ter taxas de erros maiores em ambiente real, e nenhum dos trabalhos sugere uma possível correção para este problema. Além disso, não há qualquer informação sobre a implantação do protótipo ao longo do rio, sobre como deveriam ser instalados de forma a ter maior durabilidade (como a utilização de alguns componentes a prova d'agua de baixo custo), etc. Os trabalhos também não trataram da alimentação energética dos mesmos e não houve a utilização de protocolos mais adequados para informações em tempo real de redes de sensores, como o MQTT.

Neste sentido, nenhum dos trabalhos listados entrega um protótipo disponível para uso real e distribuído. Há a demanda por uma especificação mais detalhada que possa ser seguida e possibilite a fácil implantação, uso e manutenção em ambiente real. Também se carece de dados pós-instalação de sistemas de monitoramento de baixo custo para verificar se realmente os sistemas estão aptos ao uso e se os alertas estariam contribuindo com a redução de danos.

2.3 Referencial Técnico

2.3.1 Sensores de nível

Existem diversos tipos de sensores que podem ser utilizados para medição de nível de rio, destacando-se o sensor ultrassônico, o sensor de radar e o sensor de pressão.

Os sensores ultrassônicos podem determinar níveis de água através da emissão de ondas sonoras e do cálculo de quanto tempo leva para essas ondas atingirem o alvo e retornar ao sensor. Não há a necessidade de contato com os fluídos que se deseja medir, são instalados acima do nível destes (PULTAR, 2020). A temperatura do ambiente e a humidade podem afetar a velocidade do som, sendo o primeiro o fator mais importante, entretanto, em condições atmosféricas não há uma grande variação (SCIENCEDAILY, 2020).

Os sensores de radar usam ondas eletromagnéticas para emitir a leitura do nível da água, calculando o nível com base em quanto tempo o pulso do radar demora para

retornar após o envio. Geralmente os sensores deste tipo são mais caros que outros sensores de nível. Entretanto, as ondas enviadas podem penetrar em coisas que podem interferir nas medições, como por exemplo, vapor e neblina (PULTAR, 2020).

O sensor de pressão mede a pressão/peso da água que está acima dele e converte a energia dessa pressão dos fluidos em um sinal de saída entre 4-20mA. É importante notar que os transdutores de pressão precisam estar no fundo dos rios para medições mais precisas e que estarão em contato direto com os líquidos que estão monitorando. Assim, a água deve ser limpa, caso contrário sujeiras e lama poderão danificar o sensor (PULTAR, 2020).

Cada tipo de sensor tem suas características e funcionarão melhor em determinados cenários. Por fim, para este trabalho, decidiu-se utilizar o sensor do tipo ultrassônico.

Sensores industriais de nível de rio podem ser bem caros. Verificou-se que o Serviço Autônomo de Água e Esgoto São Carlos (SAAE), por exemplo, utiliza um sensor de ultrassom industrial para medir nível de rio e tanque que custa, sozinho, cerca de R$ 6.000,00.

2.3.2 Comunicação de Dados

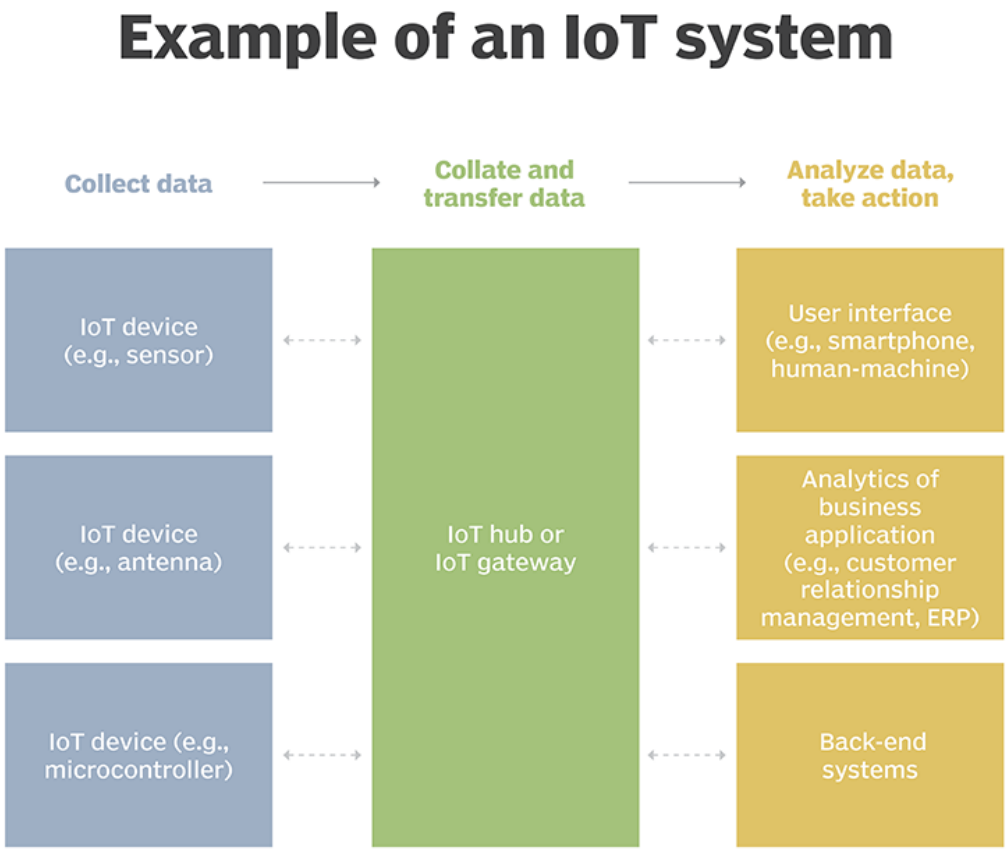
É possível utilizar diversas formas de comunicação para aplicações em Internet das Coisas, em especial as de radiofrequências, como Lora, SigFox, dentre outras. Porém, neste trabalho, optou-se pelo uso da rede de telefonia móvel (GRPS/2.5G), pois já se encontra massivamente disponível no Brasil e não necessita de nenhuma infraestrutura adicional de estação base, cabeamento, etc.

2.3.3 Arquitetura IOT

Internet of Things (IOT) pode ser definido como um sistema interconectado de diferentes tipos de dispositivos computacionais, sejam eles computadores, celulares, sistemas embarcados, relógios de pulso, casas, carros, televisores, geladeiras, equipamentos médicos, etc. onde há a transferência dos dados coletados por esses

dispositivos pela Internet para servidores na “nuvem”. Geralmente, os dispositivos que são construídos com essa capacidade são chamados de dispositivos inteligentes, *smarts* (ROUSE, 2020) (GOLDENBERG, 2020). A Figura 9 ilustra esses conceitos.

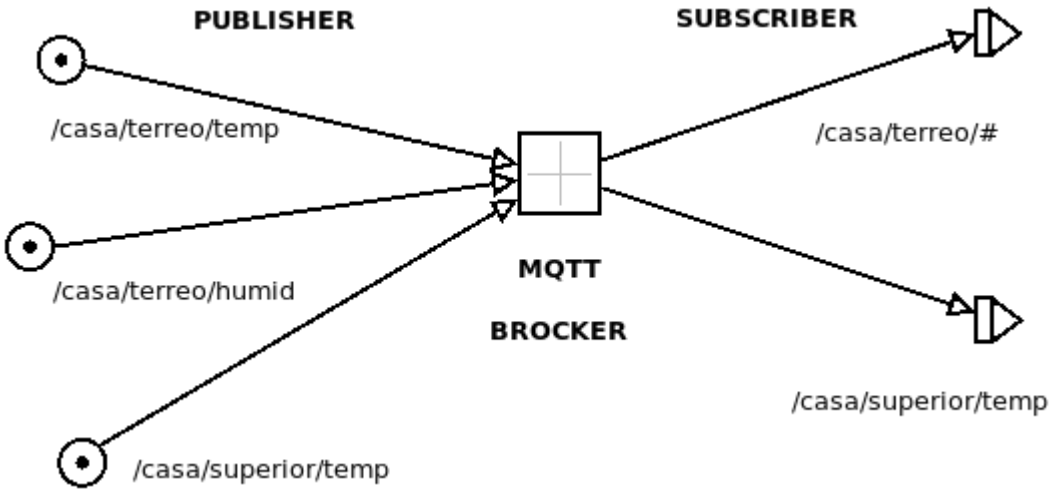
Figura 9 - Exemplo de sistema IOT



Fonte: (ROUSE, 2020)

MQTT é um protocolo de comunicação que foi projetado para ser extremamente leve e tornou-se ideal para projetos *machine-to-machine* (M2M) e de *Internet of Things* (IOT), onde a há a necessidade de envio de pequenos pacotes de dados de locais remotos – onde pode não há disponibilidade de alta largura de banda de rede - aliado a um baixo consumo de energia. Seu modelo *publish/subscribe,* permite que haja uma eficiente distribuição de informações de um publicador para vários receptores sem ter que estabelecer uma conexão direta com cada um deles (MQTT, 2020).

Figura 10 - Exemplo do modelo Publish/Subscribe



Fonte: IFPR WIKI[6]

Como pode-se notar na Figura 10, nessa arquitetura há publicadores (*publishers*) que enviam as informações para tópicos específicos que obedecem a uma hierarquia de dados. Cada nível dessa hierarquia é separado por uma barra “/”, de forma similar a organização das informações, como um sistema de arquivos (LIGHT, 2020). Por exemplo, o tópico “casa”, pode ter as informações de temperatura e humidade publicadas em “casa/temperatura” e “casa/humidade”, e para acessar todas as informações dentro da hierarquia, como “casa”, basta utilizar um jogo da velha “#” e assinar o tópico “casa/#”. Pra utilizar um tópico não é preciso criá-lo previamente, basta publicar nele. Clientes (*subscribers*) podem receber as mensagens desejadas assinando aos tópicos correspondentes (LIGHT, 2020).

O MQTT conta com três níveis de qualidade de serviços (QoS). O QoS define o quanto o broker/cliente tentará garantir que uma mensagem seja recebida. Quanto maior o nível de QoS, mais confiável, porém maior latência e maior necessidade de largura de banda (MQTT, 2020).

- QoS 0: O broker/cliente entregará a mensagem uma vez, sem confirmação;
- QoS 1: O broker/cliente entregará a mensagem pelo menos uma vez, com

[6] Disponível em: <http://wiki.foz.ifpr.edu.br/wiki/index.php/MQTT>. Acesso em: 28 jun. 2020

confirmação;

- QoS: 2: O broker/cliente entregará a mensagem exatamente uma vez, usando um *handshake* de quatro etapas.

2.3.4 Processamento e Armazenamento de Dados

No universo IOT, há o processamento local dos dados feito pelo microcontrolador e o processamento remoto, feito por servidor(es). O processamento remoto, também conhecido como computação em nuvem (*cloud computing*) permite que os dados coletados pelos dispositivos com menor poder de processamento e de armazenamento sejam enviados para servidores localizados em grandes datacenters e assim então são processados e armazenados. Esses servidores são de responsabilidade de empresas totalmente dedicadas a isso, trazendo mais integridade, disponibilidade e confiabilidade para a informação e reduzindo os custos (TECNOLOGIA, 2020).

Existem diferentes modalidades de *cloud computing*, que variam de acordo com o tipo de serviço oferecido. Para este projeto, utilizou-se os serviços da Digital Ocean[7], que é um *cloud* do tipo IaaS (*Infrastructure as a Service* ou Infraestrutura como serviço), pois oferecem VMs (máquinas virtuais) com liberdade de escolha de sistemas operacionais e de configurações. Nele, foi instalado o *broker* MQTT Mosquitto, o banco de dados MySQL e executado o script de salvamento em Python sob o Ubuntu 18.04.

3. MATERIAIS E MÉTODOS

3.1 Microcontrolador

A placa de 8 bits *open source* Arduino Mega 2560 vem equipada com o

[7] Disponível em: <https://www.digitalocean.com>. Acesso em: 28 jun. 2020

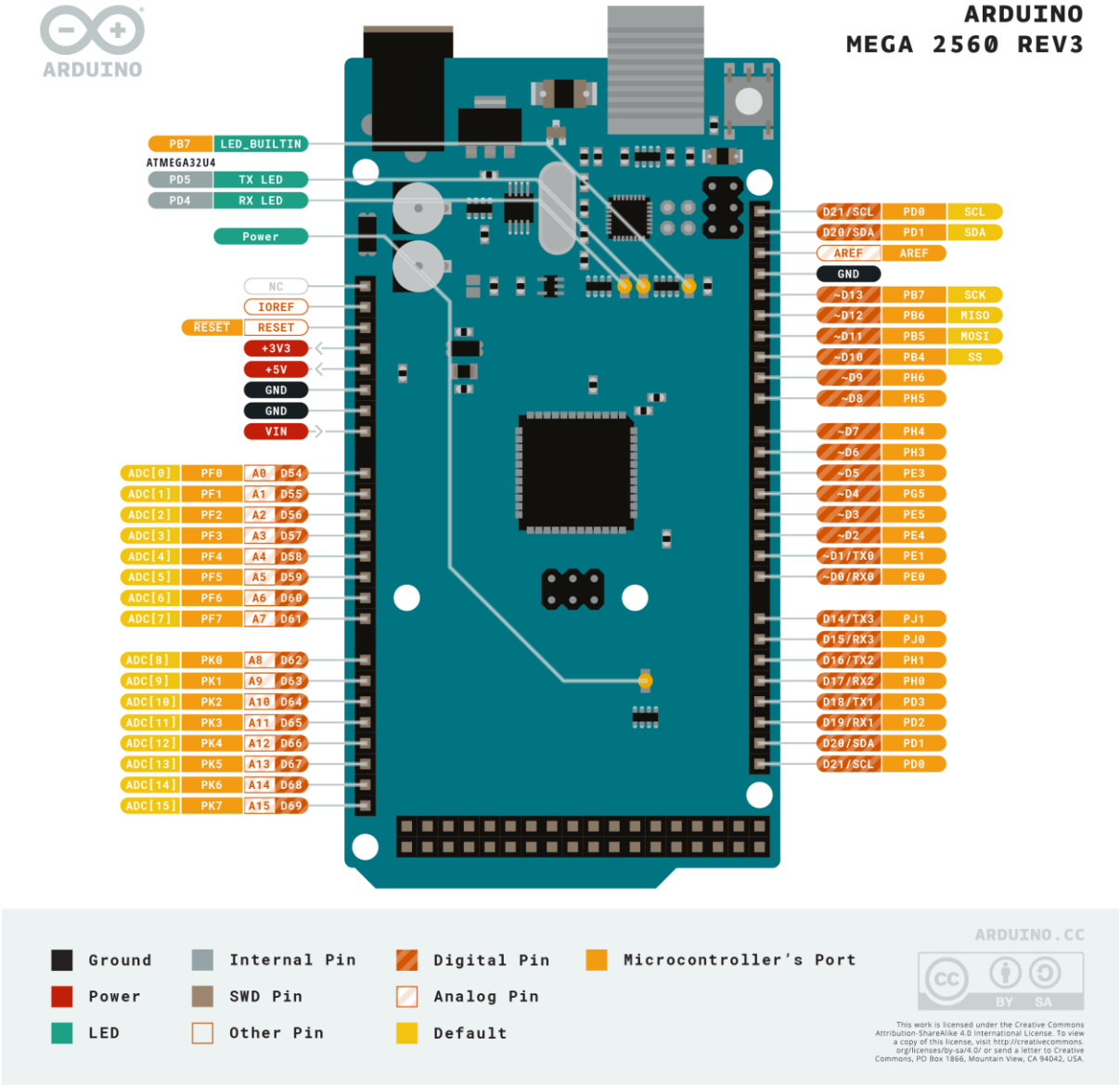
microcontrolador ATmega2560, 54 pinos digitais (dos quais 16 podem ser usados como PWM), 16 pinos analógicos e 4 portas seriais. É possível alimentá-la energeticamente através da porta USB do computador, ou um adaptador AC-DC ou uma bateria (ARDUINO STORE, 2020). A Tabela 1 mostra mais algumas informações da placa.

Tabela 1 - Especificações Técnicas do Arduino Mega

Microcontrolador	ATmega2560
Tensão operacional	5V
Tensão de entrada (recomendado)	7-12V
Tensão de entrada (limite)	6-20V
Pinos de E/S digitais	54 (dos quais 15 fornecem saída PWM)
Pinos de entrada analógica	16
Corrente DC por pino de E/S	20 mA
Corrente DC por pino de 3.3V	50 mA
Memória *flash*	256 KB, dos quais 8 KB usados pelo gerenciador de inicialização
SRAM	8 KB
EEPROM	4 KB
Velocidade de c*lock*	16 MHz
Comprimento	101.52 mm
Largura	53.3 mm
Peso	37 g

Fonte: ARDUINO STORE, (2020)

Figura 11 - Diagrama de Pinagem



Fonte: ARDUINO STORE, (2020)

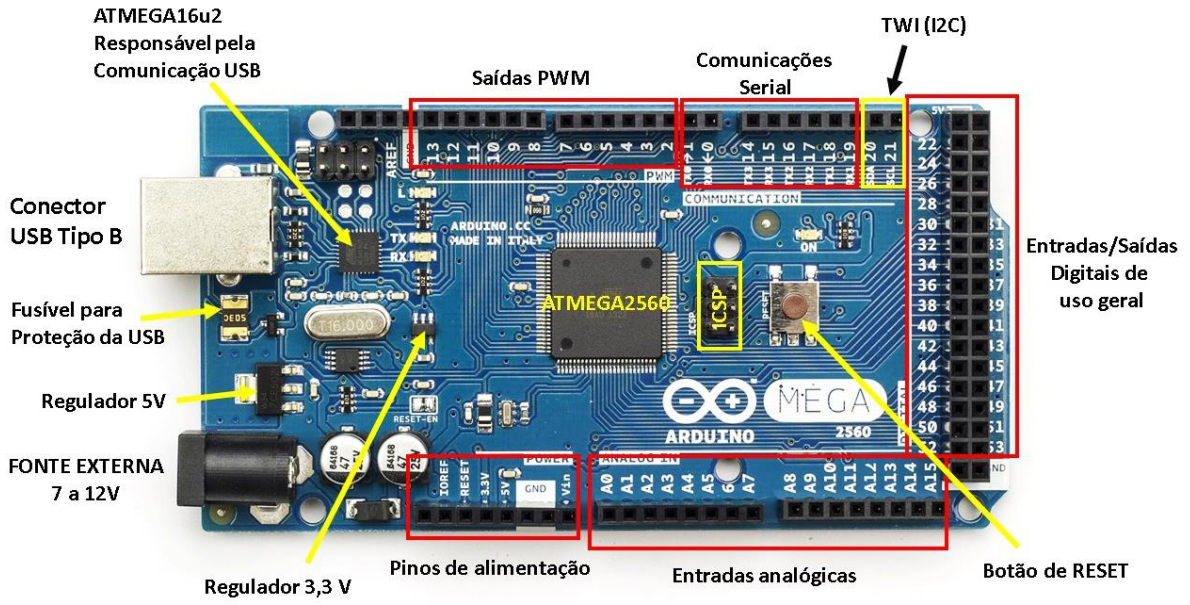
A Figura 11 mostra uma visão geral do Arduino MEGA 2560 e a Figura 12 destaca alguns de seus recursos. A plataforma Arduino além do hardware, também é composta por um software *open source* para desenvolvimento, o Arduino IDE[8]. Com a placa Arduino conectada ao computador por cabo USB pode-se usar a linguagem de programação C/C++ para descrever o comportamento dos componentes, compilá-

[8] Disponível em: <https://www.arduino.cc/en/Main/Software>. Acesso em: 13 mai. 2020.

lo e posteriormente realizar o *upload* do código compilado para a placa. Uma vez que a transferência do código foi feita para a placa não há mais dependência do computador: o Arduino irá funcionar contanto que esteja conectado a uma fonte de alimentação (FBS, 2020).

Esta placa microcontroladora de prototipação eletrônica foi escolhida para este projeto devido a alguns fatores como: baixo custo; ambiente multiplataforma (compatível com Windows, Mac Os X e Linux) de desenvolvimento rápido; utilização ativa por uma comunidade grande e colaborativa que compartilha novos conteúdos e descobertas, facilitando a obtenção de informações para realização de novos projetos; e acessibilidade, pois é fácil comprá-la em lojas especializadas no Brasil e no mundo (FBS, 2020).

Figura 12 - Arduino Mega 2560 e alguns de seus recursos.



Fonte: Site Embarcados[9]

A escolha da versão Mega 2560 - ao invés de sua versão mais famosa, o Arduino Uno - foi, principalmente, em virtude da presença de 4 portas seriais em hardware. Este projeto utiliza um módulo de GSM/GPRS que se comunica com o

[9] Disponível em: <https://www.embarcados.com.br/arduino-mega-2560/>. Acesso em: 12 mai. 2020.

microcontrolador através de comandos AT, que são enviados e recebidos através de portas seriais UART (Transmissor Universal Assíncrono). O Arduino Uno conta apenas com uma porta serial em *hardware*, o que dificultaria a elaboração do projeto, porque que a comunicação da placa Arduino com o computador utilizaria essa única porta serial. Haveria como alternativa nesse caso a implementação de serial em software, entretanto há um desempenho menor e maior probabilidade de erros de comunicação do que as implementadas em hardware, estas então, devem ser priorizadas.

A outra razão foi decorrente da maior quantidade de pinos de entrada e saída presentes neste modelo, necessários para a conexão com a placas adicionais - também conhecidas como *shields* - que podem ser acopladas ao Arduino, adicionando-o novas funcionalidades. Essas shields podem, por exemplo, adicionar funcionalidades como comunicação *Bluetooth*, *WiFi*, *Ethernet, display* LCD, etc.

3.2 Módulo GSM/GPRS

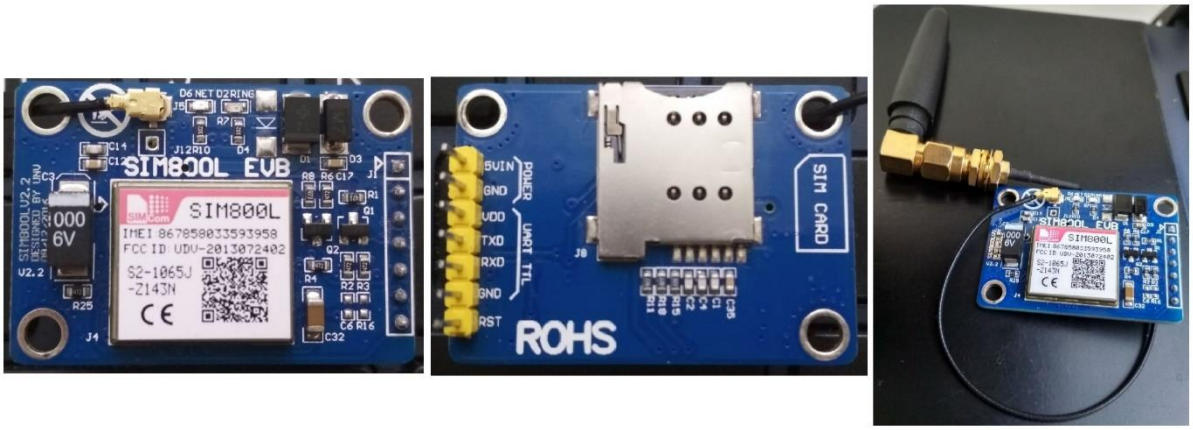
O SIM800L V2, que pode ser visualizado na Figura 13, é um módulo para comunicação através de GSM e GPRS (2G) *quadri-band* que torna possível a um microcontrolador o acesso à rede de telefonia móvel, podendo usufruir da transmissão de voz, SMS e de dados com baixo consumo de energia. Muito conveniente em cenários onde a conexão com a rede Wi-Fi e cabeada de Internet ou outros tipos meios, não são viáveis.

Neste projeto há a necessidade de comunicação em longas distâncias em cenários majoritariamente urbanos onde já há muito tempo uma infraestrutura de rede de celular com boa cobertura 2G. Além disso, o módulo foi escolhido devido ao seu baixo preço, confiabilidade, boa captação de sinal pela antena e popularidade, sendo fácil encontrá-lo para venda o que possibilita escalar o projeto. Além disso, hoje em dia o custo de pacotes de dados é muito acessível, mesmo para órgãos de serviço público.

Para sua utilização é exigido a inserção de um cartão SIM com serviços telefônicos ativos, sejam por exemplo, através de créditos pré-pagos ou planos, em alguma

operadora de telefonia. Neste projeto, por enquanto, houve apenas a necessidade de utilizar a funcionalidade de envio de dados.

Figura 13 - SIM800L V2



Fonte: da autora

Alguma de suas principais características técnicas são, segundo (SIM COM, 2020a) e (SIM COM, 2020b):

- *Quadri-band*: GSM 850 MHz, EGSM 900 MHz, DCS 1800 MHz, PCS 1900 MHz. Podendo pesquisar as quatros frequências automaticamente;
- Antena Externa;
- Alimentação de entrada de 5V com disponibilidade de corrente com disponibilidade de picos de 2A;
- Controle via comandos AT (3GPP TS 27.007, 27.005 e SIMCOM);
- Temperatura de operação de -40 °C a +85 °C;
- Transferência de dados por GPRS com velocidade de *download* e *upload* máxima de 85.6 kbps
- Porta serial suporta taxa de transmissão de 1200 bps a 115200 bps.
- Recursos de software: Protocolo TCP / UDP incorporado, FTP/HTTP

3.3 Sensor Ultrassônico

O módulo de sensor ultrassônico a prova d'água JSN-SR04T-2.0 é capaz de medir distancias correspondentes a ausência de obstáculos e detectar a presença destes obstáculos. Ele possui 3 modos de funcionamento. Para este projeto ele foi utilizado no modo 1. Neste modo, o sensor tem o funcionamento quase idêntico a outro popular

modelo de sensor ultrassônico barato, o HC-SR04, mas com a vantagem de ser ideal para medições em ambientes úmidos e molhados. Na Figura 14, há a foto deste módulo.

Figura 14 - JSN-SR04T-2.0

Fonte: da autora

Algumas de suas características técnicas estão presentes na Tabela 2 abaixo.

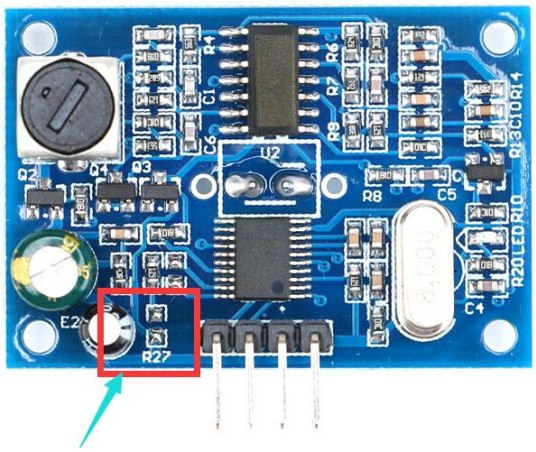
Tabela 2 - Especificações Técnicas do JSN-SR04T-2.0

Tensão operacional	DC 3.0 - 5.5V
Corrente de trabalho	Menos do que 8mA
Frequência de emissão acústica	40KHz
Distância máxima	600cm
Distância mínima	20cm
Precisão	+ - 1 cm
Resolução	1mm
Ângulo de Medição	75 graus
Entrada de sinal de *trigger*	1,10uS pulso alto TTL
Saída de sinal de *echo*	2, a porta serial para enviar instruções 0X55

Pinos	3 - 5.5V (alimentação positiva) TRIG (RX) ECHO (TX) GND (alimentação negativa)
Temperatura de operação	-20 ° C a + 70 ° C

No modo 1, que se fez uso, o ponto de soldagem R27, destacado na Figura 15, está aberto, sem nenhum tipo de resistor soldado (o que é fundamental para os modos de funcionamento 2 e 3 que não serão abordados aqui, pois não serão utilizados).

Figura 15 - Seleção do modo de trabalho



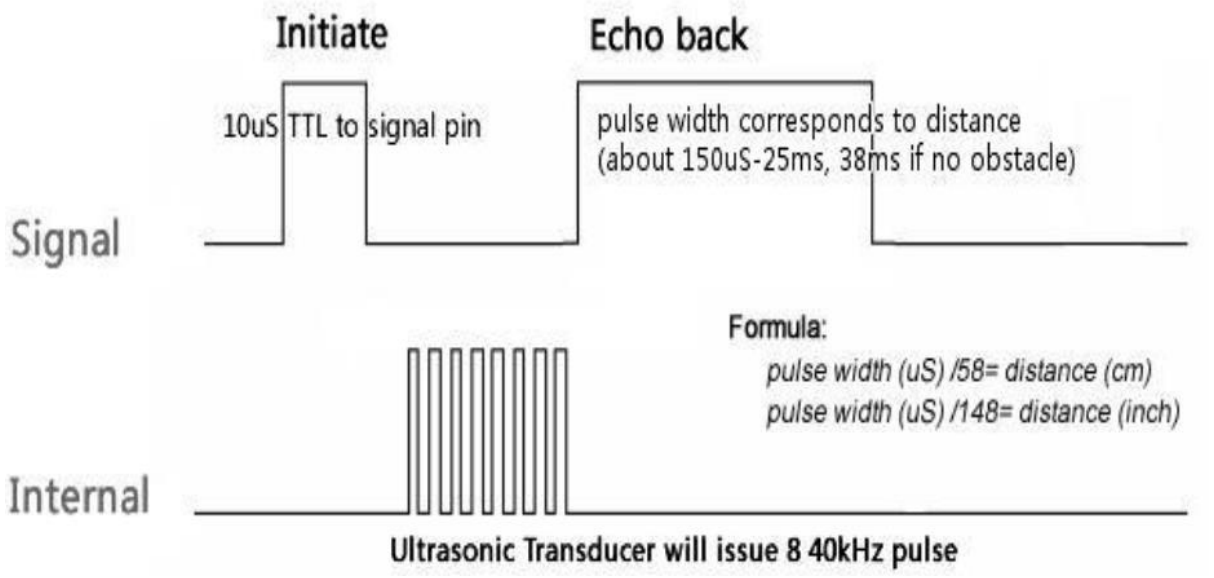
Fonte: da autora

O modo 1 tem como princípio de trabalho:

1. Envio de sinal alto de no mínimo 10µs no pino TRIG;
2. Envio pelo módulo de 8 ondas quadradas de 40khz com identificação automática de retorno;
3. O sinal de retorno é lido no pino ECHO que produz um nível alto. O tempo em nível alto corresponde a duração desde o envio do sinal até o horário de retorno. Para medir a distância, fazemos (tempo em nível alto * velocidade do som (340 m/s)) / 2;
4. O módulo recebe o gatilho após a medição da distância e se não puder receber o sinal de retorno, o pino ECHO voltará para nível baixo após 60ms, marcando o

fim da medição, seja ela bem sucedida ou não.

Figura 16 - Diagrama de temporal do sensor ultrassônico



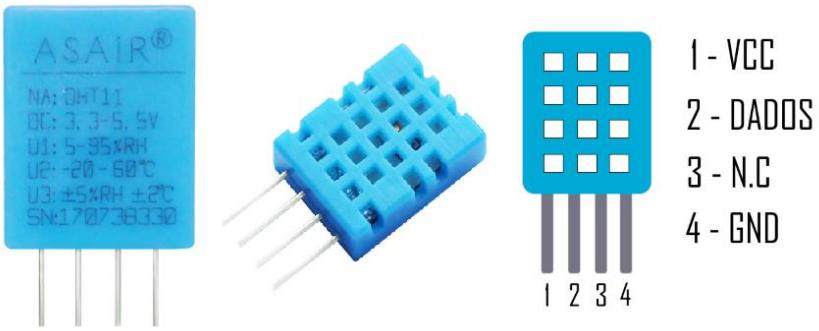
Fonte: (JAHANKIT, 2020)

Como é possível ver na Figura 16, um pulso ultrassônico curto é transmitido no momento inicial e refletido por um objeto. O sensor recebe este sinal e o converte em um sinal elétrico. O próximo pulso pode ser transmitido quando o eco desaparecer. Este tempo é chamado de período do ciclo. O período do ciclo recomendado não deve ser inferior a 50 ms. Se uma largura de pulso trigger de 10µs é enviado ao pino de sinal, o módulo ultrassônico emitirá oito sinais ultrassônicos de 40kHz e detectar o eco de volta. A distância medida é proporcional à largura do pulso de eco e pode ser calculada pela fórmula: tamanho do pulso (µs) / 58 = distancia (em cm). Se nenhum obstáculo for detectado, o pino de saída emitirá um sinal de nível alto de 38ms. (JAHANKIT, 2020)

3.4 Sensor de Temperatura e Umidade.

O sensor de temperatura e humidade DHT11 ASAIR faz medidas de temperatura entre -20 e 60 graus Celsius e humidade entre 5 e 95%. É um módulo de baixo custo, estável à longo prazo, com resposta rápida, forte capacidade anti-interferência, saída digital e calibração precisa. Na Figura 17, pode-se ver a aparência deste sensor e sua pinagem. Sua interface de comunicação de dados é feita por apenas um pino.

Figura 17 - Sensor DHT11



Fonte: da autora

Alguns de seus parâmetros técnicos (ASAIR, 2020):

- Tensão de Alimentação: DC 3.3 – 5.5V;
- Precisão de medição para humidade: +-5%;
- Precisão de medição para temperatura: +-2 ºC;
- Resolução de Temperatura: 0.1 ºC;
- Tempo de resposta de humidade: 6s;
- Tempo de resposta de temperatura: 10s.

3.5 Fonte de Alimentação

O conversor AC-DC LUXE-P-60-12 é uma fonte chaveada *bivolt* (110V-220V / 50Hz-60Hz) que estabiliza a tensão de saída a partir do controle de corrente por chaveamento do circuito. Sua saída é de 12V com corrente de 5A, com uma potência de 60W. Essa fonte foi escolhida devido a necessidade de picos de 2A de corrente para alimentar o módulo SIM800L.

Neste projeto, devido ao seu contexto urbano, decidiu-se aproveitar da rede de distribuição de energia elétrica pública que há ao longo dos rios da cidade de São Carlos como fonte de alimentação para o protótipo em implantação. Assim sendo, não há necessidade de troca de baterias, tornando a manutenção muito mais fácil e menos necessária. Em uma versão futura, pretende-se utilizar painéis solares. Na Figura 18 é possível visualizar este módulo.

Figura 18 - AC-DC LUXE-P-60-12

Fonte: da autora

3.6 Regulador de Tensão

O módulo regulador de tensão LM2596, presente na Figura 19, é um conversor DC-DC capaz de converter uma entrada de entre 3,2V até 40V em uma saída ajustável entre 1,2V e 37V com uma corrente máxima de 3A (INSTRUMENTS, 2020). Esse ajuste é feito através do giro do parafuso do potenciômetro do circuito integrado com o acompanhamento do valor de tensão de saída com por exemplo, um multímetro, até se obter o valor desejado. Neste projeto, a tensão de saída do regulador foi configurada em 5V, servindo como alimentação direta para o Arduino e todos os outros módulos acoplados.

Figura 19 - LM2596

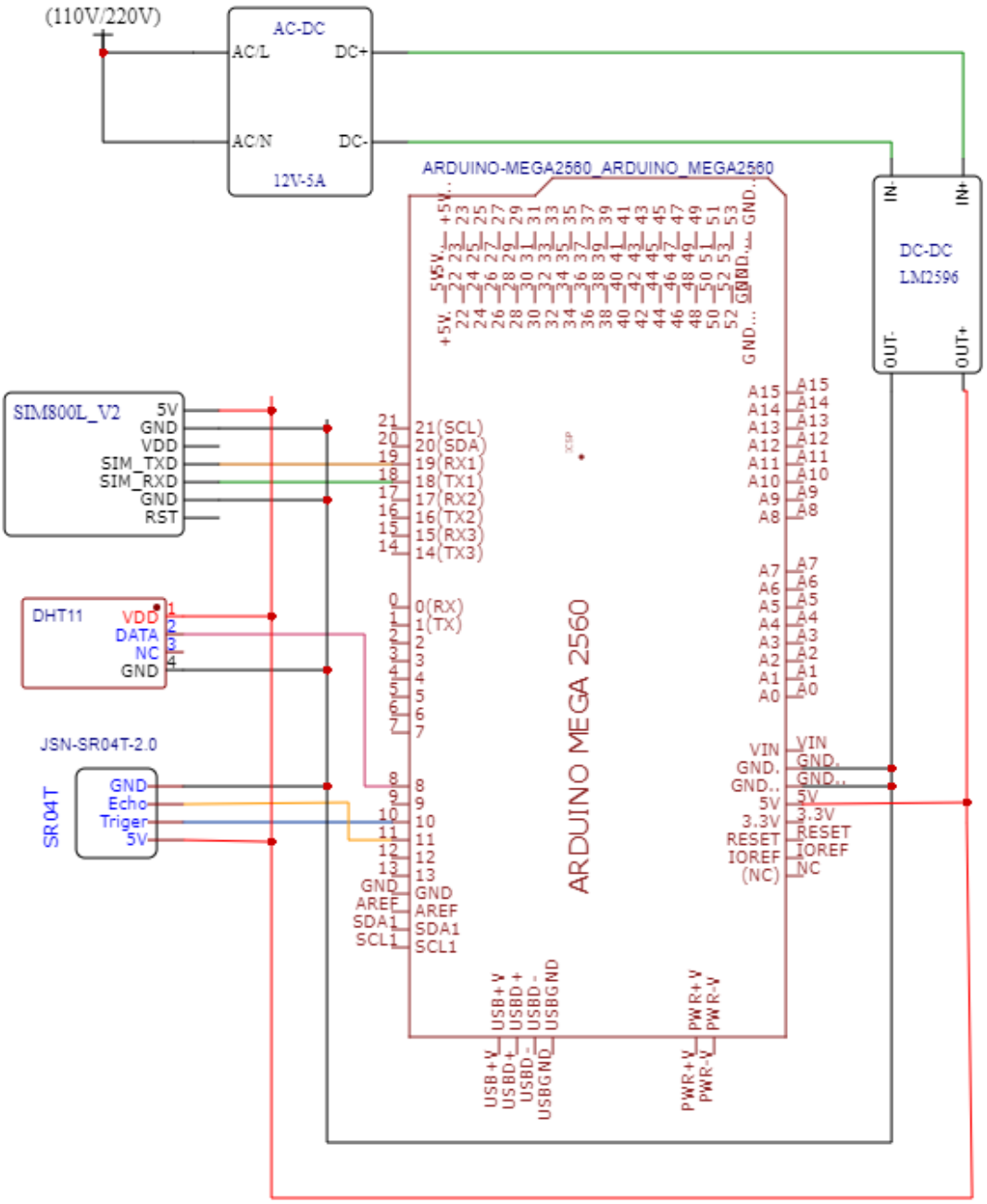
Fonte: da autora

3.7 Protótipo

A Figura 20 mostra o diagrama esquemático do circuito desenvolvido neste trabalho. A energia de entrada do conversor de corrente alternada em corrente contínua provém da rede de distribuição elétrica, que pode ter como intervalos de tensão de 110V à 120V ou 200V à 220V. Ela é, então, convertida em 12V em corrente contínua e passada pelo regulador de tensão LM2596 que a transforma em 5V, sendo distribuída através de uma placa de circuito universal (de soldagem) para todos os elementos do sistema em suas respectivas entradas 5V e GND.

Com as conexões de alimentação feitas, restam as conexões de sinais. O módulo JSN-SR04T-2.0 tem seu pino TRIG e ECHO conectados respectivamente aos pinos 10 e 11 do Arduino Mega. O SIM800L V2, tem seus pinos UART seriais TXD e RXD conectados a porta serial1 da placa, respectivamente aos pinos 19 (RX1) e 18 (TX1). E por fim, o DHT11 ASAIR tem seu único pino de dados conectado ao pino 8.

Figura 20 - Diagrama esquemático do circuito



Fonte: da autora

3.8 Arquitetura

O arranjo do sistema foi pensado com base nos modelos de comunicação de sistemas distribuídos cliente-servidor e *publish-subscribe*. Os dados coletados em tempo real das estações de monitoramento de uma determinada cidade são distribuídos através do protocolo MQTT, enquanto o *software* de administração e monitoramento web e

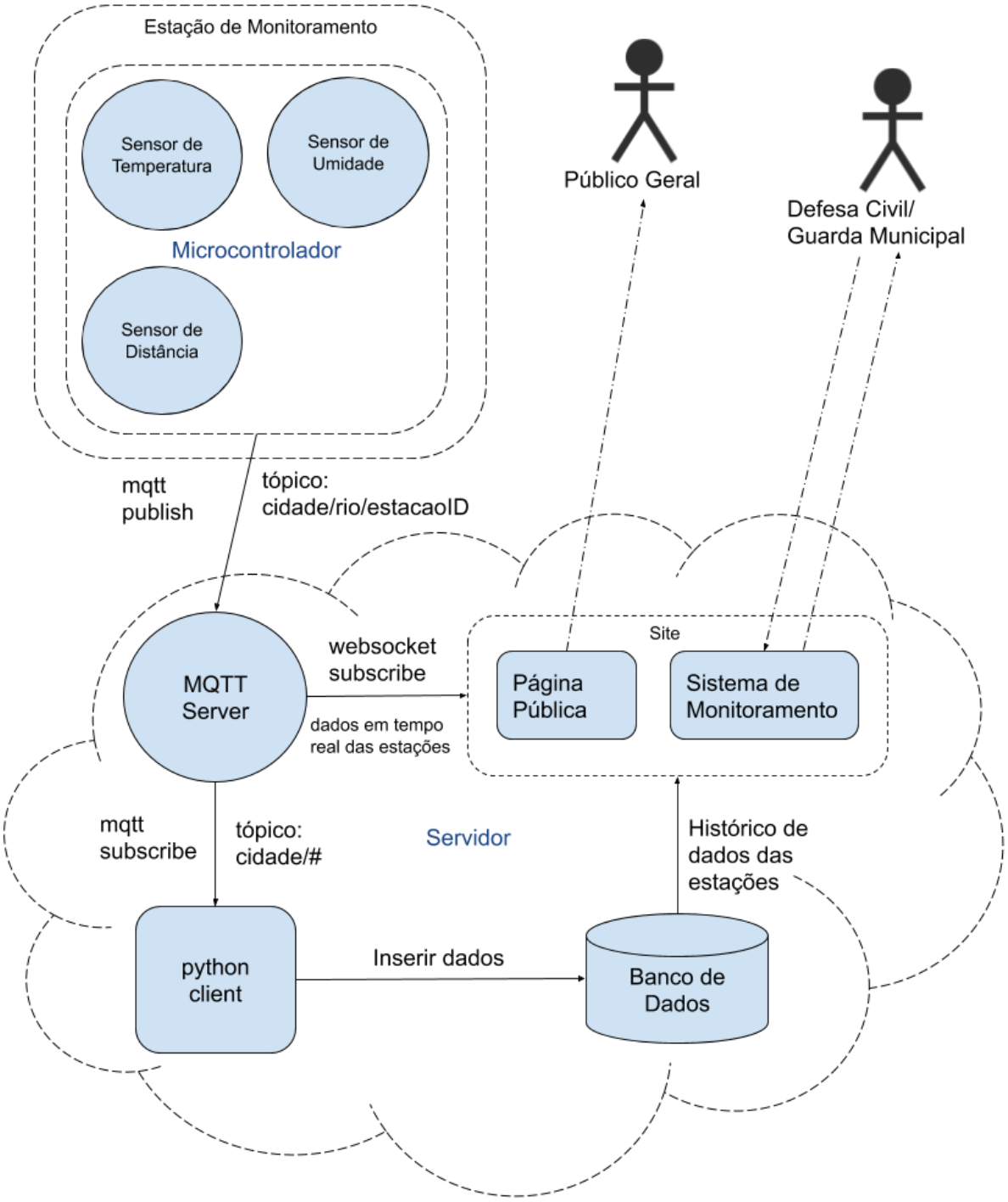
seu banco de dados são transmitidos através de HTTP/HTTPS.

Cada estação de monitoramento pertence a uma cidade, a um rio e tem um ID único que o identifica. Assim sendo, ele posta seus dados no tópico “cidade/rio/estacaoID” no servidor MQTT (antigamente denominado *broker*). Com essa estrutura, é possível filtrar os dados ao se tornar consumidor de um tópico. Por exemplo, para receber os dados de todas as estações de uma determinada cidade, basta se inscrever no tópico “cidade/#”; seguindo a mesma lógica, para receber todos os dados de um determinado rio, “cidade/rio/#”.

Os dados de todas as estações, por exemplo, de uma cidade, são inseridos em um banco de dados relacional através de um *script* em Python que roda em *background* no mesmo servidor alocado para funcionamento do Servidor MQTT. Desta forma, com os históricos de dados dos sensores ao longo do tempo armazenados será possível visualizá-los através de um site com um mapa pelo público geral, assim como os dados em tempo real de cada estação via *websocket* do *broker*. Além disso, em um sistema de monitoramento com *login* para usuários específicos da Defesa Civil e/ou da Guarda Municipal, será possível cadastrar estações, usuários, visualizar gráficos, cadastrar níveis de alerta e enviá-los, etc. Isso que foi explicado está representado no diagrama da Figura 21.

É importante ressaltar que devido ao curto tempo de elaboração deste trabalho não foi viável integrar todas as partes da arquitetura, como o site, o que será mencionado mais adiante na seção de trabalhos futuros.

Figura 21 - Arquitetura do Sistema de Monitoramento Completo

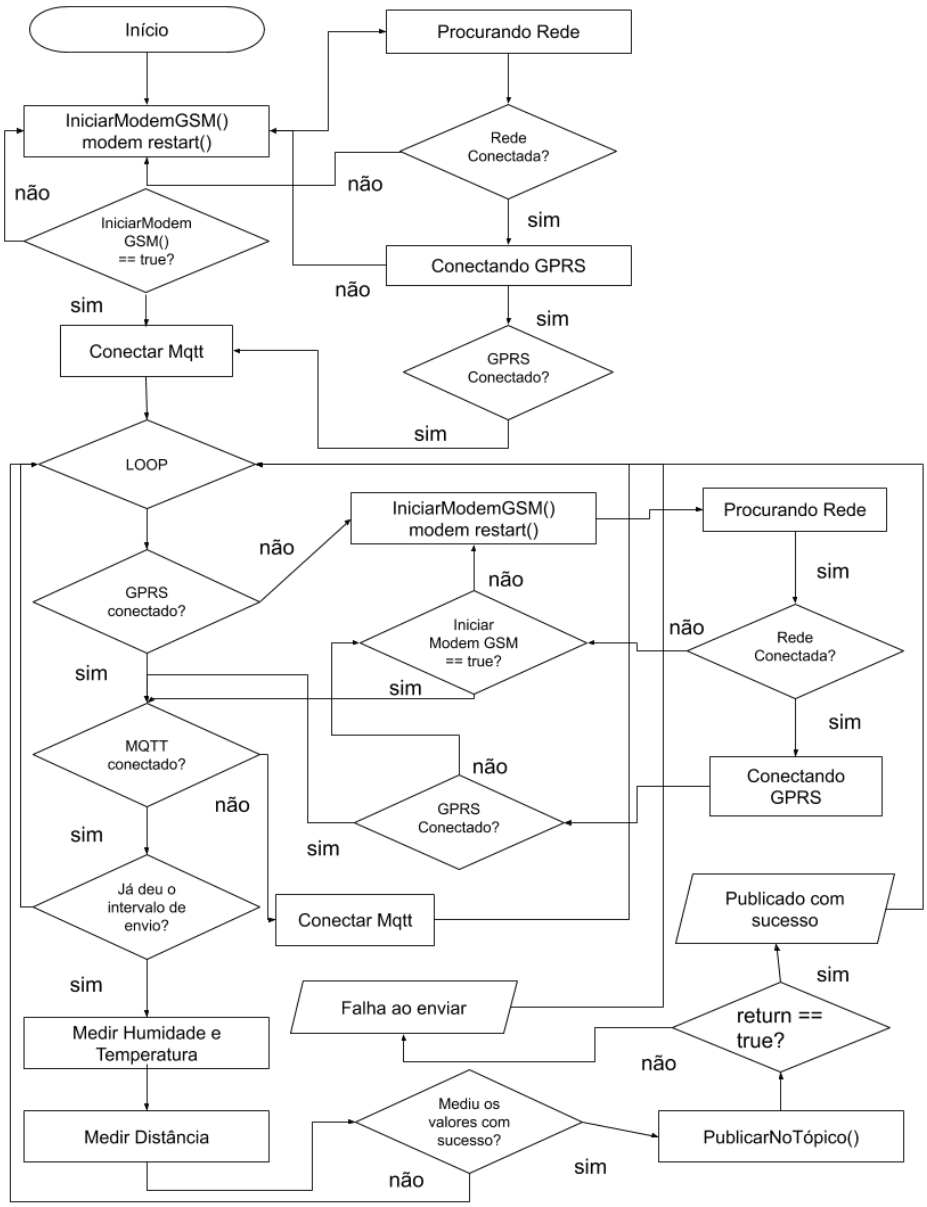


Fonte: da autora

3.9 Lógica da Estação de Monitoramento

Na Figura 22 é ilustrada através de um diagrama a lógica simplificada geral de funcionamento do programa (foram ocultados alguns procedimentos) escrito em linguagem C aplicado ao microcontrolador do Arduino Mega. O código está disponível no Apêndice A, no fim deste documento.

Figura 22 - Lógica Simplificada de Funcionamento



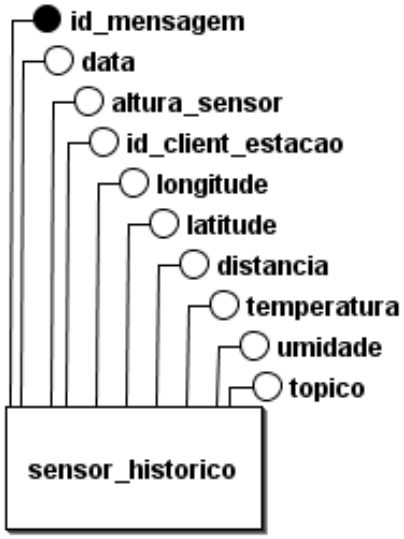
Fonte: da autora

3.10 Banco de Dados

Para construção do banco de dados utilizou-se sistema de gerenciamento de banco de dados (SGBD) de código aberto MySQL[10] na sua versão 8.0.19. Criou-se uma tabela para arquivamento dos dados de histórico das estações de monitoramento. O código de criação da tabela encontra-se à disposição na seção de Apêndice C.

Nas Figuras 23 e 24, pode-se ver os diagramas representativos desta tabela.

Figura 23 - Diagrama Conceitual



Fonte: da autora

Figura 24 - Diagrama Lógico

Fonte: da autora

Para que os dados enviados por MQTT sejam salvos de forma persistente é necessário que um *client script* em Python seja executado no servidor. O código desse script encontra-se no Apêndice B deste trabalho. O comando abaixo, usado no sistema operacional Ubuntu 18.04 faz com que o programa seja executado em *background.* A saída do programa fica salva no arquivo `nohup.out`.

```
nohup python3 -u ./mqtt_to_db.py &
```

[10] Disponível em: <https://www.mysql.com>. Acesso em 25 jun. 2020

3.11 Servidor MQTT Mosquitto

O *broker* MQTT escolhido para o projeto foi o Eclipse Mosquitto[11] devido a sua leveza, disponibilidade multiplataforma, boa documentação e código aberto. A versão do protocolo usada foi a 3.1.1. Os comandos abaixo foram utilizados em um servidor com sistema operacional Ubuntu 18.04. Após a instalação do servidor MQTT configurou-se um usuário e senha de acesso:

```
mosquitto_passwd -b /etc/mosquitto/passwd username password
```

Abriu-se o arquivo de configurações padrão:

```
sudo nano /etc/mosquitto/conf.d/default.conf
```

Feito isso, adicionou-se a seguintes linhas no arquivo `default.conf` afim de desabilitar conexões anônimas sem autenticação e de informar ao servidor onde a senha criptografada está armazenada.

```
allow_anonymous false
password_file /etc/mosquitto/passwd
```

Em conexões não criptografadas a porta 1883 é usada por padrão para comunicação. Depois de instalado, para iniciá-lo operando como um serviço em *background* basta utilizar:

```
systemctl start mosquitto
```

[11] Disponível em: <https://mosquitto.org>. Acesso em 25 jun. 2020

O código da estação de monitoramento utiliza a biblioteca PubSubClient[12] para se conectar ao Mosquitto e as mensagens são enviadas com QoS 0, ou seja, a mensagem é enviada uma única vez sem confirmação de chegada.

O ideal seria que fossem enviadas com QoS 2, o que significa que todas as mensagens seriam enviadas exatamente uma vez através de um processo de quatro etapas de confirmações. Entretanto, neste caso, seria necessário armazenar temporariamente as mensagens que não fossem capazes de serem enviadas no momento, o que pode, segundo (O'LEARY, 2020), ser complicado em um hardware de pouca memória e sem um mecanismo padrão de persistência de dados.

[12] Disponível em: <https://pubsubclient.knolleary.net>. Acesso em 25 jun. 2020

4. RESULTADOS E DISCUSSÃO

4.1 Troca de módulo GSM/GPRS

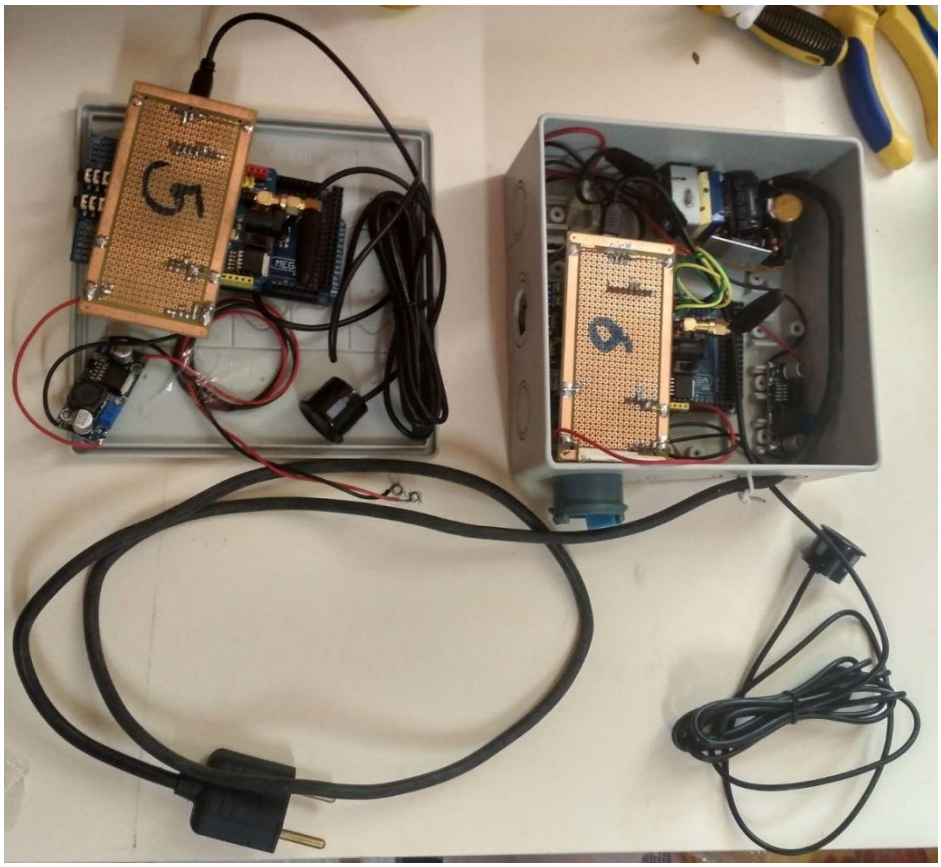
No início do projeto foi utilizado o SIM900, que é o modelo antecessor do SIM800 e inclusive mais caro que este, porém, devido a fatores técnicos de má alimentação, não se conseguia fazer o SIM800L funcionar, o que foi posteriormente solucionado. Durante este período, entretanto, descobriu-se mais fatores desencorajantes ao uso do SIM900, como o fato de este sensor não se auto reiniciar com o reinício do Arduino, o que fazia com que houvesse a necessidade de sempre enviar comandos AT para verificar se o módulo estava ligado para então enviar um sinal de inicialização, o que não ocorre com o SIM800L. Este, inicia-se assim que recebe a alimentação adequada.

Figura 25 - SIM900

Fonte: da autora

Na Figura 26, pode-se ver dois protótipos de duas estações de monitoramento com o uso do SIM900, que posteriormente foram trocados.

Figura 26 - Estações com uso do SIM900

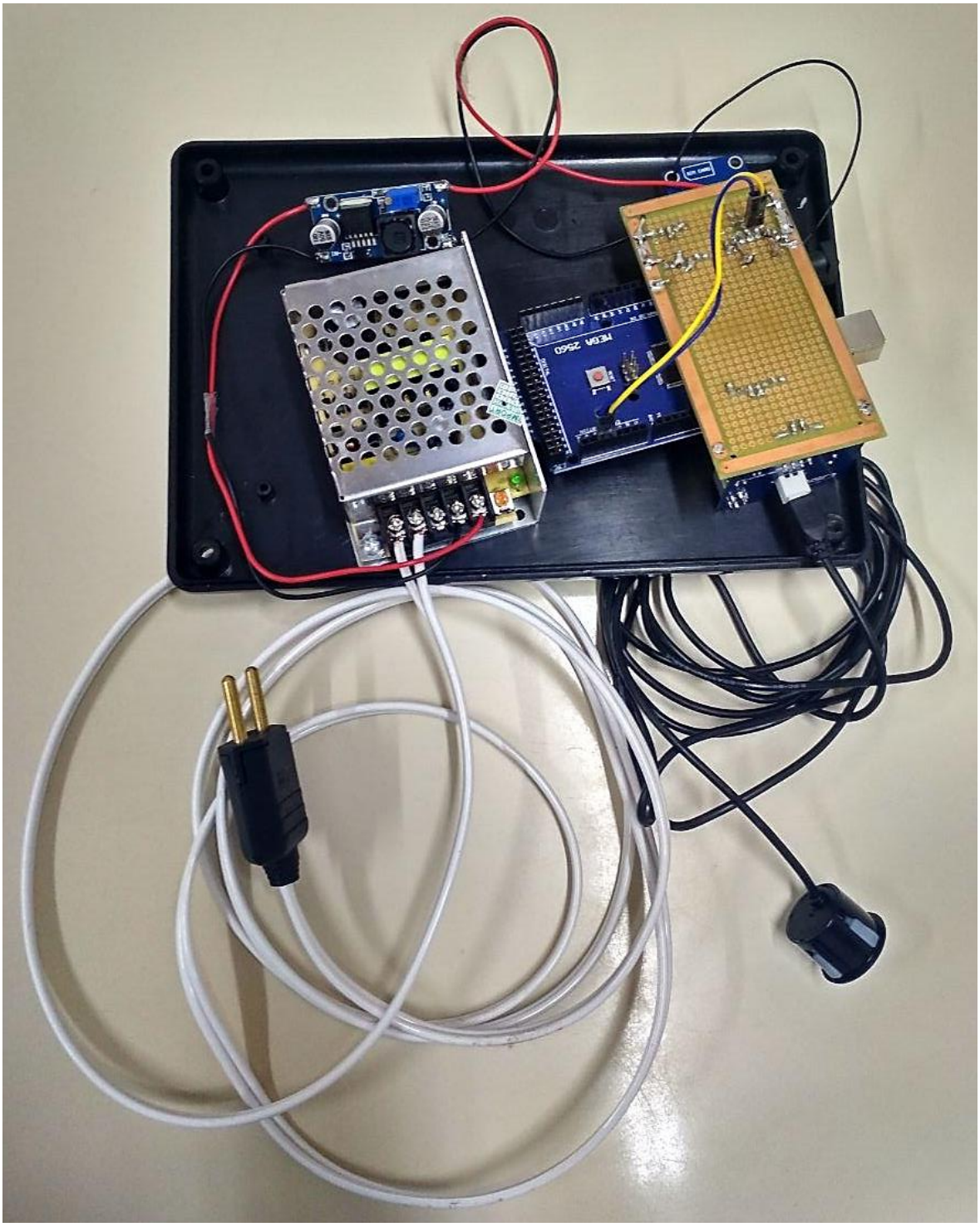


Fonte: da autora

4.2 Montagem e teste do protótipo

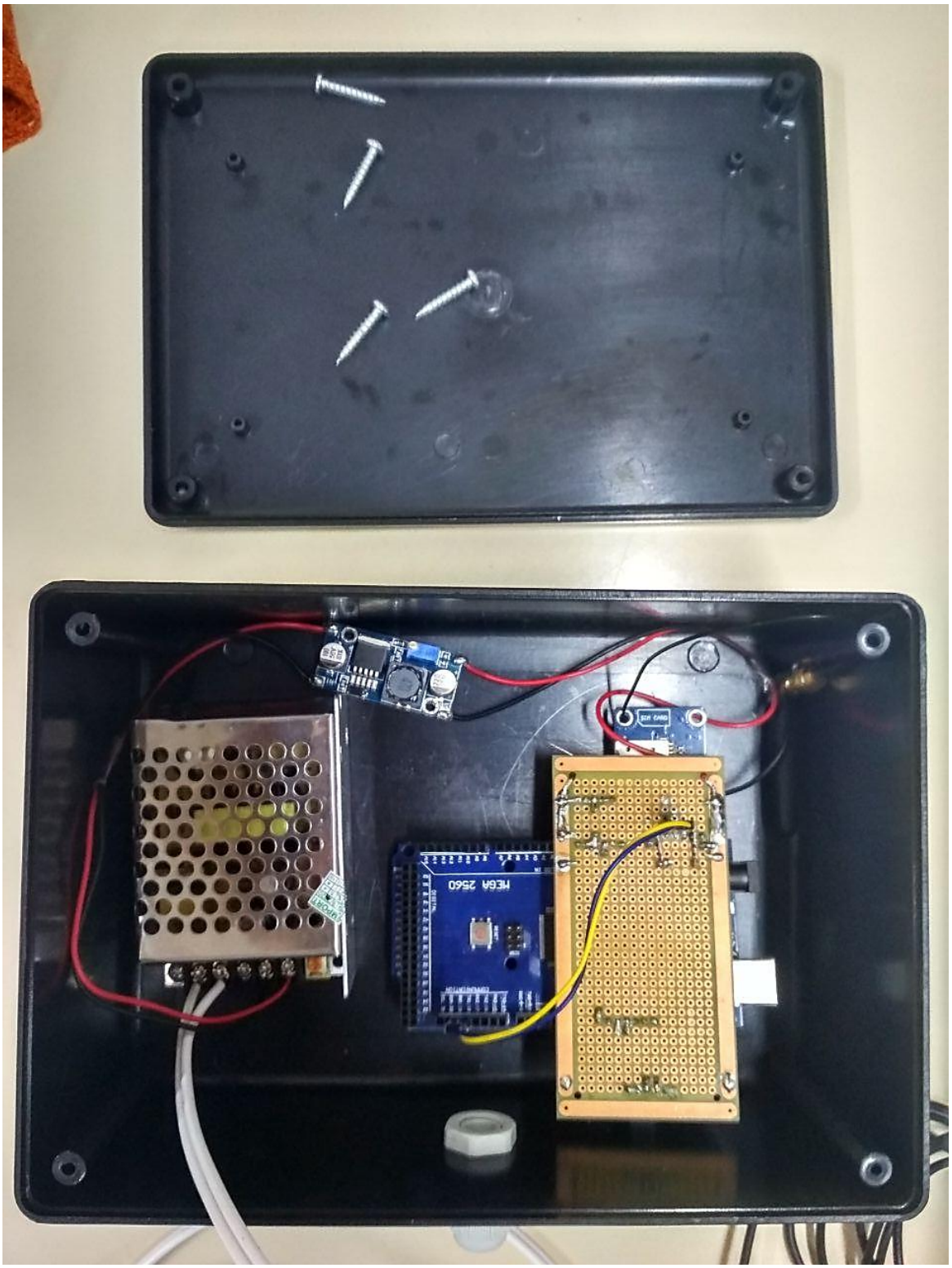
Seguindo as orientações do diagrama esquemático do circuito e utilizando uma caixa de proteção com passa fio, tem-se o que é visualizado nas Figuras 27, 28, 29 e 30 abaixo.

Figura 27 - Protótipo



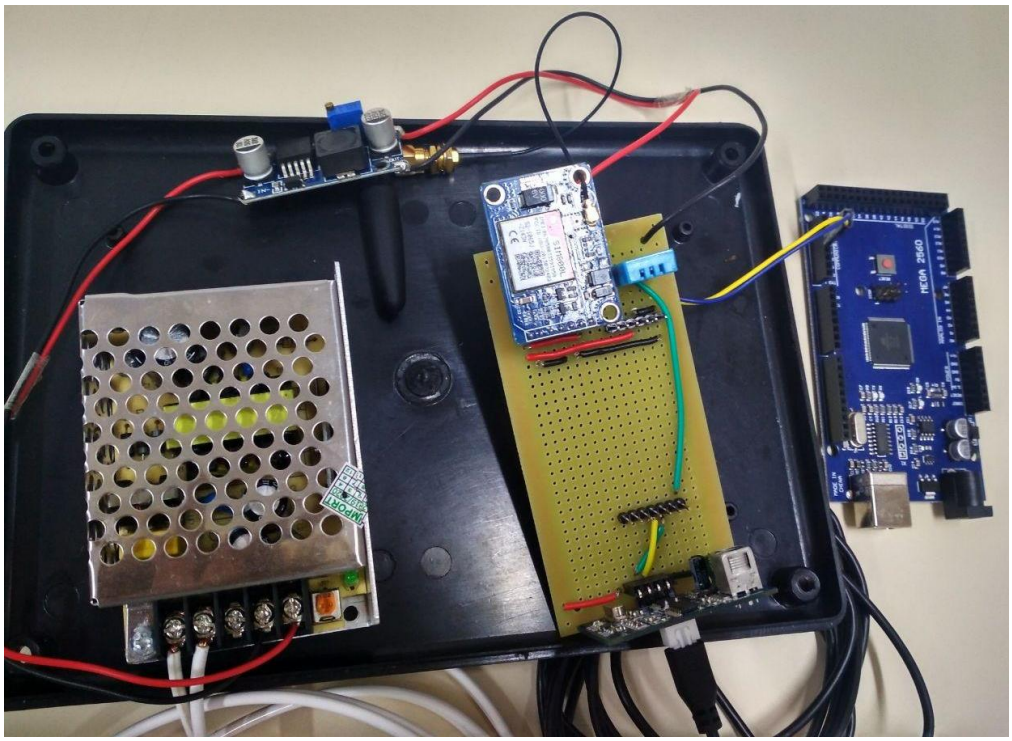
Fonte: da autora

Figura 28 - Protótipo dentro da caixa de proteção



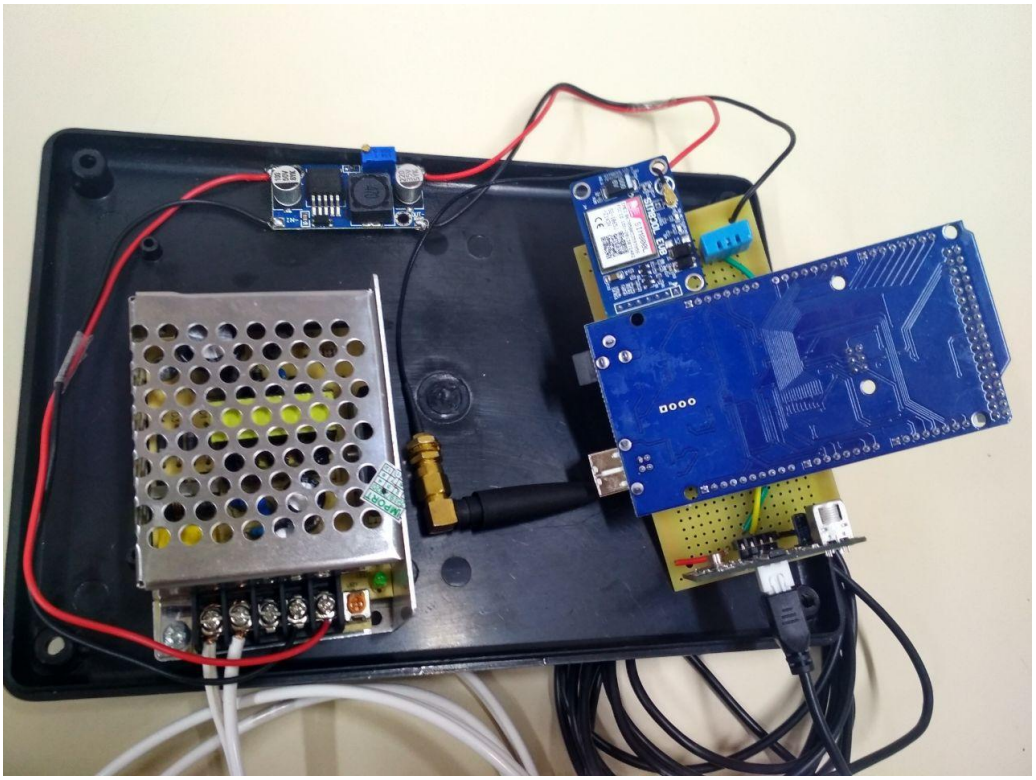
Fonte: da autora

Figura 29 - Visão do verso da placa universal do protótipo



Fonte: da autora

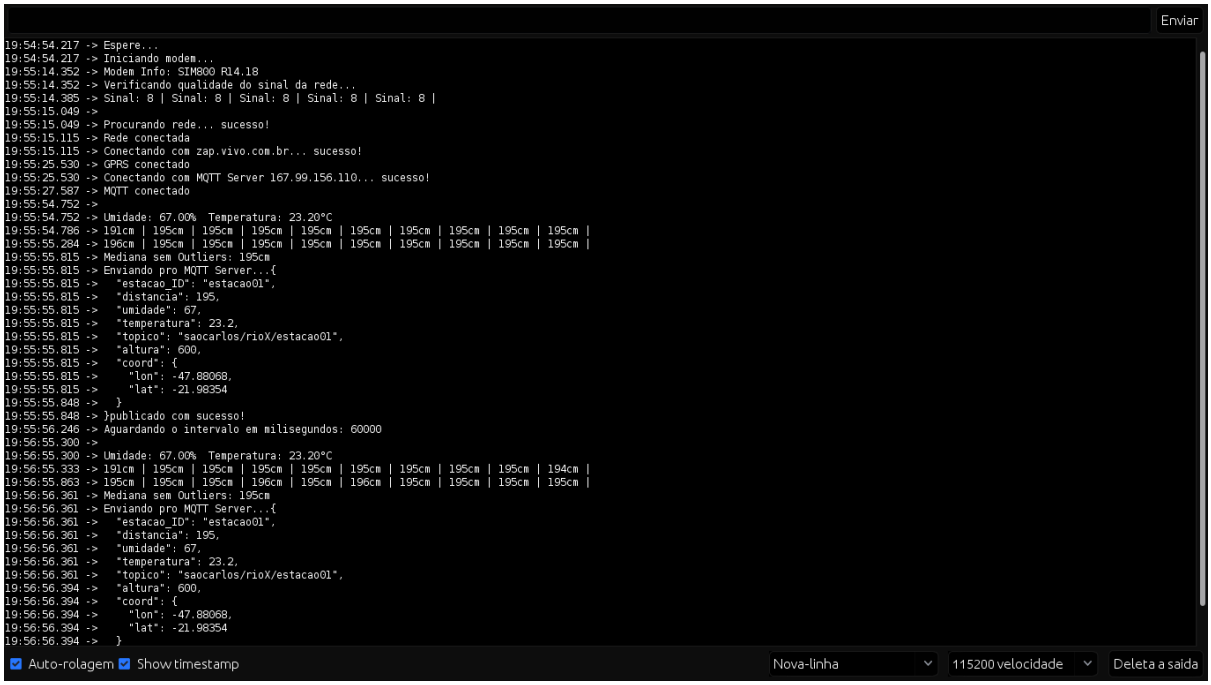
Figura 30 - Verso do protótipo



Fonte: da autora

Na Figura 31, tem-se um exemplo da visualização das informações de execução do código do microcontrolador no *Serial Monitor* da Arduino IDE.

Figura 31 - Debug do código no Serial Monitor da Arduino IDE

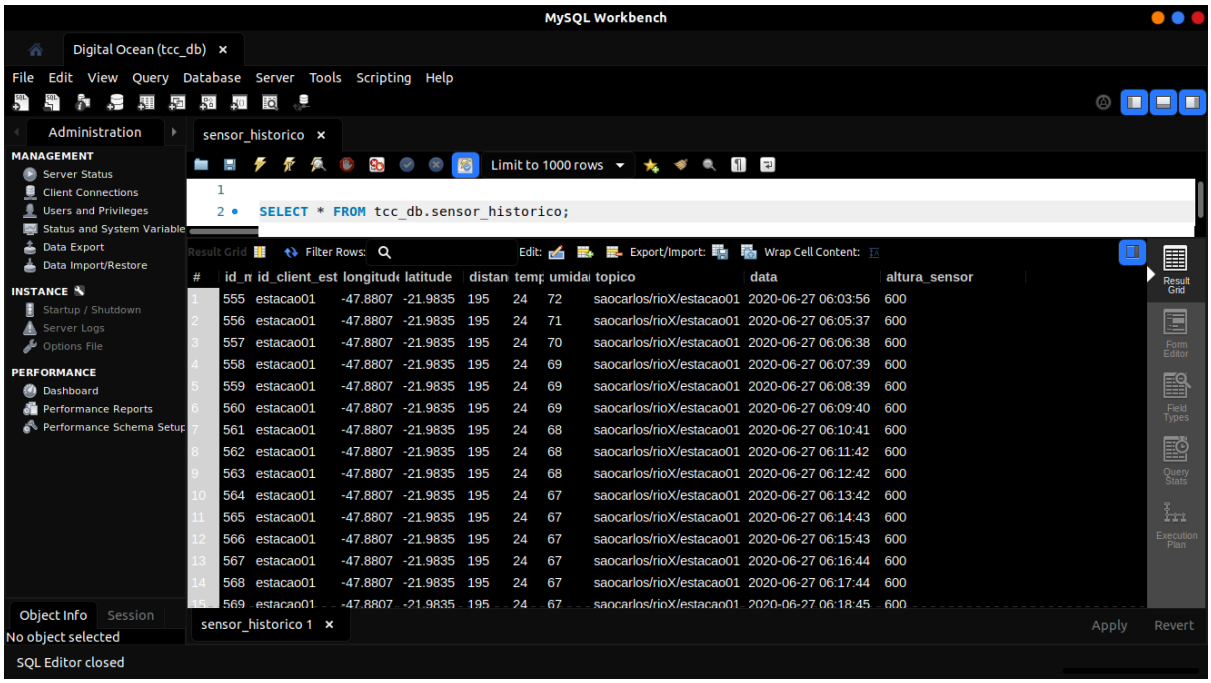


Fonte: da autora

Utilizando o MySQL Workbench[13] pode-se conectar ao servidor de banco de dados remoto e visualizar os dados armazenados, como pode ser visto na Figura 32.

[13] Disponível em: <https://www.mysql.com/products/workbench/>. Acesso em 26 jun. 2020

Figura 32 - Gerenciamento do Banco de Dados pelo MySQL Workbench

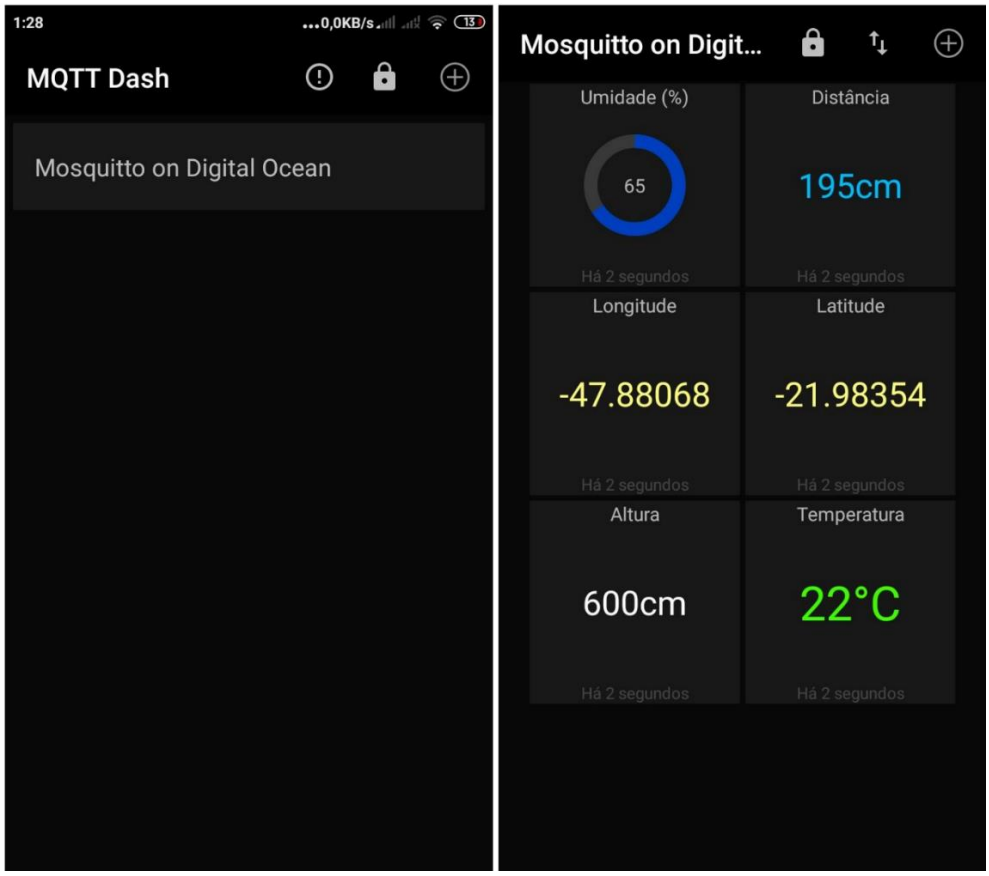


Fonte: da autora

Também é possível utilizar, por exemplo, a título de teste, um aplicativo como cliente para visualizar os dados enviados por MQTT em tempo real. Para isso, utilizou-se o MQTT Dashboard[14]. Na Figura 34, estão detalhadas as informações de configuração para mostrar cada informação enviada por uma estação de monitoramento.

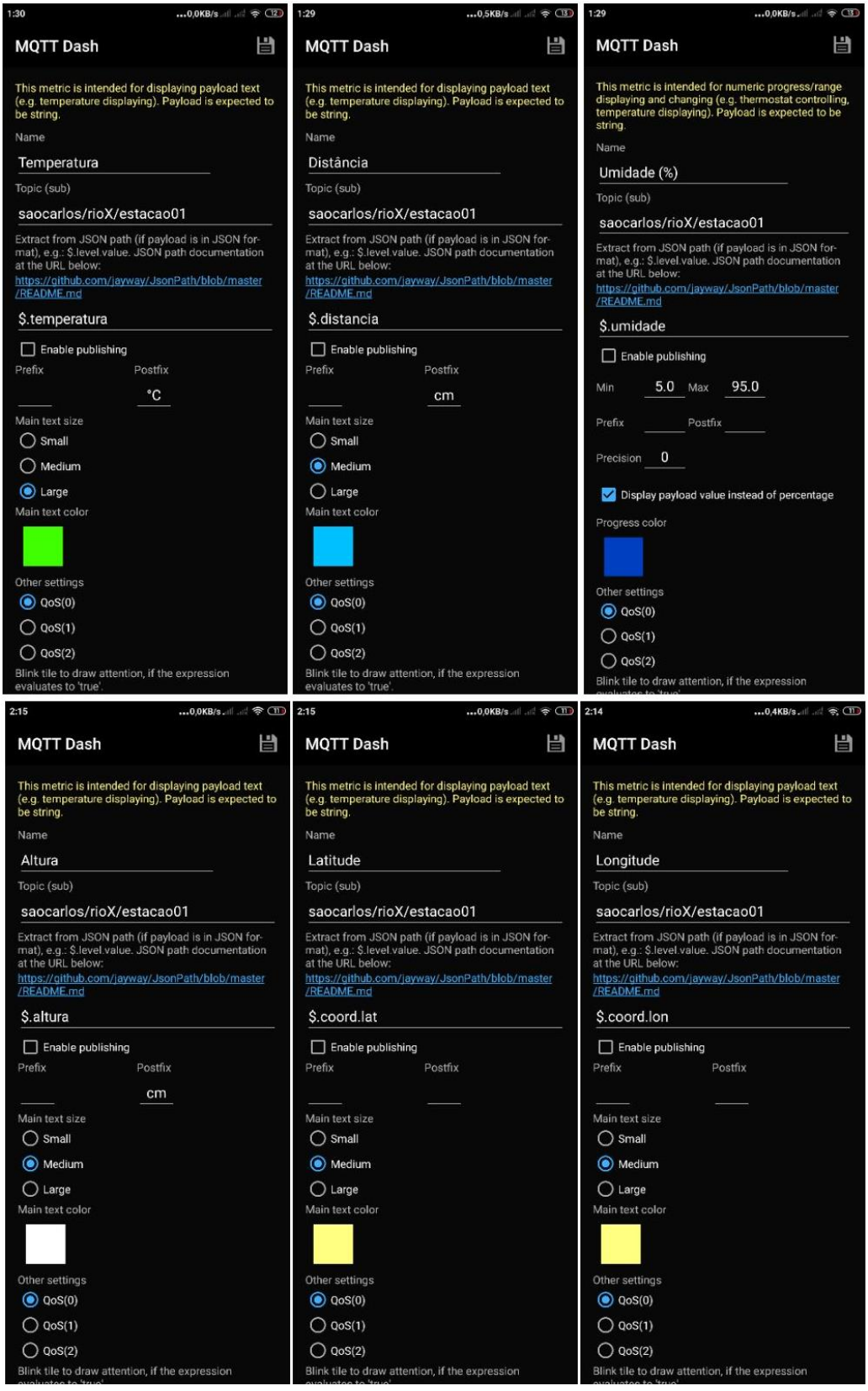
[14] Disponível em: <https://play.google.com/store/apps/details?id=net.routix.mqttdash&hl=pt_BR>. Acesso em 26 jun. 2020

Figura 33 - Exibição de dados no aplicativo MQTT Dash



Fonte: da autora

Figura 34 - Configurações MQTT Dash

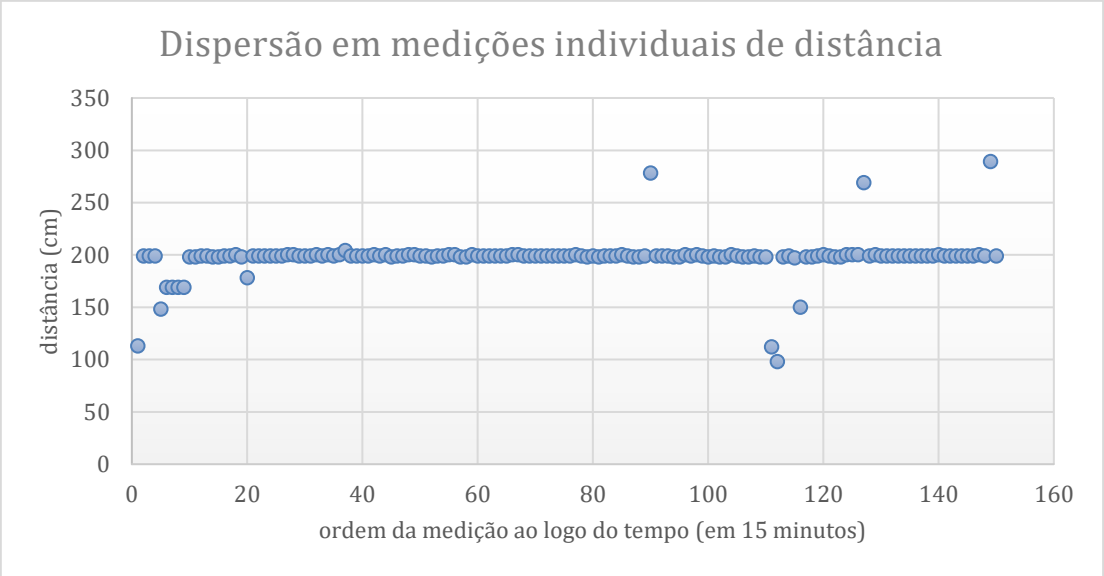


Fonte: da autora

4.3 Filtragem do sinal do sensor ultrassônico

Durante o desenvolvimento deste trabalho, notou-se que o sensor de ultrassom apresentava alguns números bem discrepantes dos valores corretos, como pode ser exemplificado pelo gráfico presente na Figura 35. Esses dados discrepantes, entretanto, não foram notados nos dados de temperatura e humidade que sempre se mostraram relativamente estáveis com apenas uma única medição.

Figura 35 - Gráfico de dispersão em medições individuais de distância de sensor ultrassônico



Fonte: da autora

Visando melhorar a precisão da medição, considerando que diversos fatores aleatórios externos podem causar a perda ou desvio das ondas enviadas pelo sensor, procurou-se coletar uma amostra de 10 medições e fazer a média das coletas individuais de distância. Entretanto, verificou-se que o valor final da distância ainda era bem afetado pelos valores discrepantes, como pode ser analisado na Tabela 3.

Tabela 3 - Dados de amostra, sua média e após filtragem durante 15 minutos de coleta

Hora	Média (cm)	"Sem" outliers (cm)	Amostra (cm)
06:00:20.765	173	169	113 199 199 199 148 169 169 169 169 198
06:01:21.400	196	198	198 199 199 198 198 199 199 200 198 178

06:02:21.931	199	199	199	199	199	199	199	199	200	200	199	199
06:03:22.594	199	199	199	200	199	200	199	200	204	199	199	199
06:04:23.102	199	199	199	200	199	200	198	199	199	200	200	199
06:05:23.636	199	199	199	198	199	199	200	200	198	198	200	199
06:06:24.140	199	199	199	199	199	199	199	200	200	199	199	199
06:07:24.647	199	199	199	199	199	199	199	199	200	199	198	199
06:08:25.252	206	199	198	199	199	199	200	199	198	198	199	278
06:09:25.892	198	199	199	199	199	198	198	200	199	200	199	198
06:10:26.524	198	198	199	198	198	200	199	198	198	199	198	198
06:11:27.163	174	198	112	98	198	199	197	150	198	198	199	200
06:12:27.655	206	199	199	198	198	200	200	200	269	199	200	199
06:13:28.358	199	199	199	199	199	199	199	199	199	199	199	200
06:14:28.889	208	199	199	199	199	199	199	199	200	199	289	199

Segundo (BAE; JI, 2019), para detectar valores extremos em dados de nível d'água medidos por sensores ultrassônicos deve-se usar um estimador robusto que seja insensível a valores extremos com grandes desvios.

Huber, Leys et al, Rousseeuw e Croux, citados por (BAE; JI, 2019), mostram que a mediana da amostra pode ser usada como um estimador robusto, pois a mediana é muito insensível aos efeitos de grandes desvios e portanto, podem minimizar o efeito da assimetria causada por valores discrepantes na distribuição de dados de nível d'agua coletados por sensores ultrassônicos. Além disso, o MAD (*median absolute deviation*), pode ser usado como um estimador de escala robusto para estimar a dispersão de dados centralizados na mediana da amostra.

Desta forma, concluiu-se que para detectar os múltiplos dados discrepantes da amostra não se deve usar o desvio padrão em torno da média, e sim o desvio absoluto em torno da mediana (MAD).

Iglewicz e Hoaglin citados por (BAE; JI, 2019), sugerem o uso do Z-score modificado.

$$Dado\ um\ conjunto\ de\ dados\ X = \{x1, x2, \dots, xn\}$$

$$MAD(X) = Med\{|X - Med\ (X)|\}$$

$$Mi = 0.6745 \times \frac{|xi - Med(X)|}{MAD(X)} < \beta$$

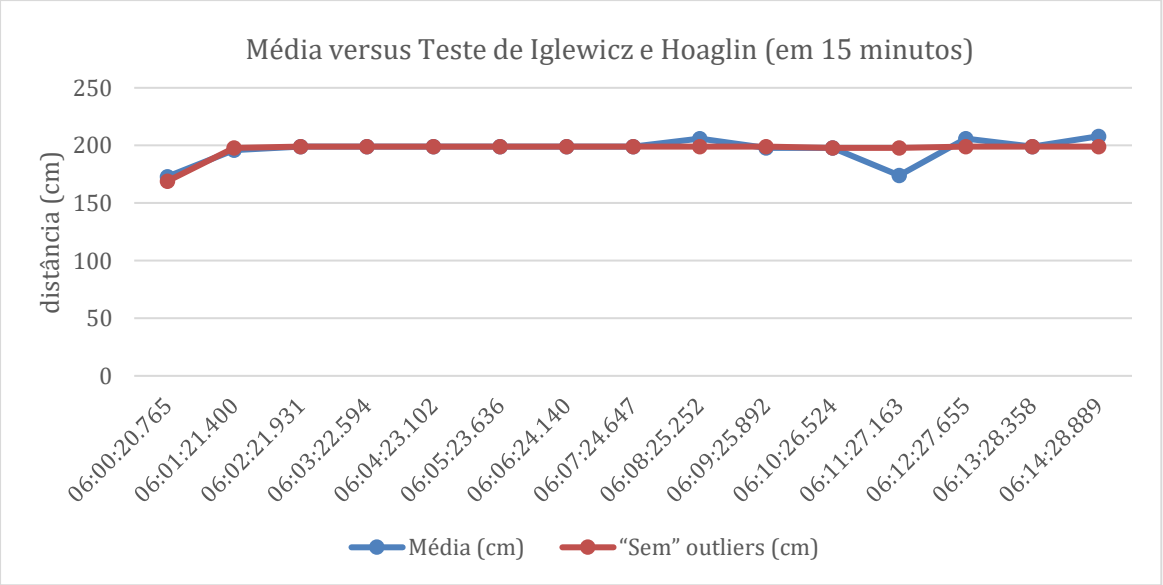
com MAD (X) denotando o desvio absoluto em torno da mediana do conjunto X, Med (X) sendo a mediana de um conjunto X, Mi sendo o Z-score Modificado e β sendo o critério de rejeição que determina se um dado de medição é um *outlier*. O número 0.6745 equivale a 75% da distribuição normal, que corresponde ao MAD da distribuição normal com um desvio padrão de 1.

Esses dois autores recomendam que os Z-score modificados com valores absolutos maiores que β = 3,5 sejam rotulados como possíveis discrepantes. Porém, para o nosso caso, utilizou-se um valor bem menos tolerante de β = 0,7. Com o conjunto final de medições sem *outliers*, foi adaptado, para este estudo, um novo cálculo de mediana finalizando a filtragem.

$$Distância = Med\left\{0.6745 \times \frac{|xi - Med(X)|}{MAD(X)} < \beta\right\}$$

Na Figura 36 é possível a comparação entre os métodos de minimização de efeitos de outliers com os dados apresentados na Tabela 3.

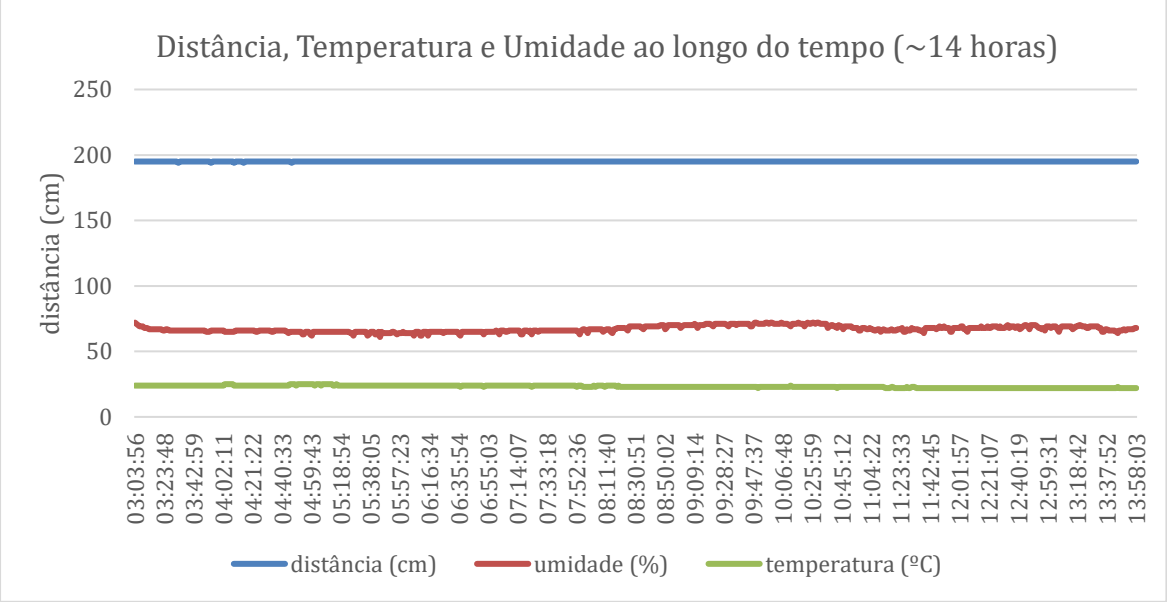
Figura 36 - Média versus Teste de Iglewicz e Hoaglin



Fonte: da autora

Afim de obter melhores resultados, como já mencionado, adotou-se um β de 0,7 e uma amostra maior, de 20 medições individuais de distância. A Figura 37 mostra o gráfico do teste final de aproximadamente 14 horas de medições de distância, temperatura e humidade. É bem perceptível na imagem que os dados do sensor de ultrassom se encontram muito mais suavizados e estáveis, praticamente sem nenhum *outlier*.

Figura 37 - Teste final de medição dos sensores da estação de monitoramento



É imprescindível que os dados de distância estejam corretos, afinal ele é o principal dado na tomada de decisão acerca dos alertas que serão implementados futuramente.

4.4 Análise de desempenho

Em um cenário de redes moveis há sempre a possibilidade de perda de sinal e consequentemente de perda de pacotes de informação. Neste sentido, um teste de desempenho foi executado com o sistema instalado na UFSCar, no dia 26 de junho de 2020. O teste constituiu em contar o número de tentativas de envio de pacotes MQTT com sucesso e com falha por erro de comunicação ou perda do sinal da rede. O resultado pode ser visto na Tabela 4.

Tabela 4 - Teste para quantificação de perda de mensagens

Tempo de teste	19 horas
Quantidade de mensagens recebidas	1125
Quantidade de mensagens esperada	1140
Quantidade de mensagens perdidas	15
Porcentagem de perda de mensagens	1,31%

4.5 Exemplo de Instalação

Uma versão de um protótipo anterior, desenvolvido sem o uso de MQTT e sem uso do SIM800L, foi instalado em São Carlos, próximo à rotatória da USP, na avenida Francisco Pereira Lopes. Apesar de não estar em completo funcionamento e de ter outro código de funcionamento, serve como exemplo de instalação, pois em breve o protótipo construído neste trabalho será instalado de forma similar. É possível ver na Figura 38 como poderia ser feita a instalação de uma estação de monitoramento às margens de um rio.

Figura 38 - Exemplo de instalação de uma estação de monitoramento

Fonte: Finéias Silva

4.6 Custos do protótipo final

Os componentes foram escolhidos com o intuito de obter uma boa eficiência e confiabilidade com baixo custo. O preço nacional de cada item foi obtido na cidade de São Carlos. É importante salientar que adquirindo os produtos diretamente da China pode haver uma redução de custos de até mais da metade do preço praticado em território nacional. A aquisição dos componentes em maior escala também pode reduzir os preços. O cálculo dos custos pode ser verificado na Tabela 5. Não foram contabilizados os custos com fios.

Tabela 5 - Cálculo dos custos do protótipo

Componente	**Preço Nacional**
Arduino Mega	R$ 108,00
Sensor DHT11 ASAIR	R$ 19,66
Conversor AC-DC 60W 12V 5A	R$ 35,53
JSN-SR04T-2.0	R$ 84,70
Regulador de tensão LM2596	R$ 13,50
SIM800L V2	R$ 89,50
Caixa de Plástico para projeto Eletrônico	R$ 73,00
Placa universal fibra perfurada 5x10cm	R$ 8,00
Total:	**R$ 431,89**

5. CONCLUSÕES

Neste estudo iniciou-se o desenvolvimento de um protótipo para monitoramento de rios utilizando tecnologias *open source* e de baixo custo visando sua futura aplicação no mundo real com simplicidade e robustez, algo que ainda falta no mercado. Sendo assim houve uma enorme preocupação em tornar o sistema estável e pouco tolerante a falhas. Para isso, foram realizados diversos testes em busca de problemas que pudessem ser melhorados.

Muito mais testes ainda terão de ser feitos, afinal, devido ao curto período de dois meses para desenvolvimento deste projeto não foi possível realizar todos os testes planejados. Novos sensores – e componentes no geral, de diferentes fabricantes e preços - podem ser futuramente testados em diversos tipos de condições climáticas com o propósito de obter um melhor custo-benefício.

A implantação de um sistema de hardware e software como este traria um enorme benefício para a sociedade, principalmente a brasileira, tendo em vista que o país sofre inúmeras perdas devido a enchentes causada por rios em regiões urbanas, onde a grande maioria não são monitorados.

Além disso, por meio deste projeto pude aprofundar meus conhecimentos em MQTT, IOT, redes, sistemas embarcados, circuitos, arquiteturas e documentações, estatística, soldagem de componentes, etc. E muito ainda poderá ser aprendido com a realização de novas funcionalidades.

5.1 Trabalhos Futuros

Devido a curto tempo disponível para a implementação deste TCC, muitas funcionalidades não puderam ser implementadas. Consta listado abaixo algumas melhorias a serem feitas:

- Integração com o Google Maps;
- Desenvolvimento do site e seu banco de dados;

- Integração do Servidor MQTT e do banco de dados com o site do sistema de monitoramento;
- Criação de uma porta *websocket* no Mosquitto sem autenticação com permissão de apenas leitura (*subscribe*) para conexão com o site e possível disponibilização de dados não sensíveis das estações de monitoramento publicamente;
- Mais segurança no tráfego de informações de autenticação com a incorporação da camada de segurança SSL/TLS no MQTT;
- Sistema de alerta através de aplicativos de mensagens, como o Telegram[15];
- Uso de Watch Dog no código para maior estabilidade;
- Implantação com energia de painel solar;
- Testagem em condições críticas;
- Melhoria do código para que dados de configuração - como localização - possam ser alterados e não declarados diretamente no código;
- Adição de cartão SD para salvamento de dados persistentes;
- Análise dos dados dos sensores ao longo do tempo;
- Melhoramento no código de funcionamento do microcontrolador;
- Melhoramento do código de detecção de outliers;
- Melhoramento da forma de salvar as informações no banco de dados.

[15] Disponível em: <https://telegram.org>. Acesso em 26 jun. 2020

REFERÊNCIAS

MAROTTI, Ana Cristina Bagatini et al. Levantamento histórico e relatos de inundações do córrego do Gregório na região central do município de São Carlos - SP. **Revista EIXO**, Brasília, DF, v. 3 n. 1, 2014. Disponível em: <https://doi.org/10.19123/eixo.v3i1.141>. Acesso em: 18 nov. 2019.

HAPSARI, Ratih Indri; ZENURIANTO, Mohammad. View of flood disaster management in Indonesia and the key solutions. **American Journal of Engineering Research**, v. 5, n. 3, 2016. p. 140-151. Disponível em: <http://www.ajer.org/papers/v5(03)/T050301400151.pdf>. Acesso em: 18 nov. 2019.

TUCCI, Carlos EM. Inundações urbanas. **Porto Alegre: ABRH/RHAMA**, Porto Alegre, RS, v. 11, 2007. Disponível em: <http://www.mpf.mp.br/atuacao-tematica/ccr4/importacao/institucional/grupos-de-trabalho/encerrados/residuos/documentos-diversos/outros_documentos_tecnicos/curso-gestao-do-terrimorio-e-manejo-integrado-das-aguas-urbanas/drenagem1.PDF>. Acesso em: 18 nov. 2019.

INTHARASOMBAT, Ouychai; KHOENKAW, Paween. A low-cost flash flood monitoring system. In: **2015 7th International Conference on Information Technology and Electrical Engineering (ICITEE)**. IEEE, 2015. p. 476-479. Disponível em: <https://ieeexplore.ieee.org/abstract/document/7408993>. Acesso em: 18 nov. 2019.

DOS REIS, João Bosco Coura. Monitoramento e alerta de inundação no município de Itajubá (MG) através de modelos matemáticos. 2014. Tese de Doutorado. **Dissertação de Mestrado em Ciências em Meio Ambiente e Recursos Hídricos**, Itajubá, MG. Disponível em: <http://www.terrama2.dpi.inpe.br/wp-content/uploads/2017/12/dissertacao_joao_unifei.pdf>. Acesso em: 18 nov. 2019.

PATIL, Sonali et al. A Real Time Solution to Flood Monitoring System using IoT and Wireless Sensor Networks. **Int. Res. J. Eng. Technol**, v. 6, n. 2, p. 1807-1811, 2019.

Disponível em: <http://www.academia.edu/download/59934295/IRJET-V6I235620190704-73858-174jm9k.pdf>. Acesso em: 18 nov. 2019.

ARDUINO STORE. **ARDUINO MEGA 2560 REV3**. Disponível em: <https://store.arduino.cc/usa/mega-2560-r3>. Acesso em: 12 mai. 2020.

FBS, Eletrônica. **APOSTILA ARDUINO**. Disponível em: <http://www.valdick.com/files/ApostilaArduinoIntroducao.pdf>. Acesso em: 13 mai. 2020.

SIM COM, a company of SIM Tech. **SIM800L (MT6261) Hardware Design V1.01**. Disponível em: <https://simcom.ee/documents/SIM800L/SIM800L%28MT6261%29_Hardware%20Design_V1.01.pdf>. Acesso em: 24 jun. 2020a.

SIM COM, a company of SIM Tech. **GSM/GPRS Module SIM800L**. Disponível em: <https://simcom.ee/documents/SIM800L/SIM800L%20SPEC170914.pdf>. Acesso em: 24 jun. 2020b.

JAHANKIT. **JSN-SR04T-2.0**. Disponível em: <https://www.jahankitshop.com/getattach.aspx?id=4635&Type=Product>. Acesso em: 24 jun. 2020.

INSTRUMENTS, Texas. **LM2596 SIMPLE SWITCHER® Power Converter 150-kHz 3-A Step-Down Voltage Regulator.** Disponível em: <https://www.ti.com/lit/ds/symlink/lm2596.pdf?ts=1593101649032>. Acesso em: 25 jun. 2020.

ASAIR. **DHT11 SIP Packaged Temperature and Humidity Sensor**. Disponível em: <http://www.aosong.com/en/products-21.html> Acesso em: 25 jun. 2020.

BAE, Inhyeok; JI, Un. Outlier Detection and Smoothing Process for Water Level Data Measured by Ultrasonic Sensor in Stream Flows. **MDPI and ACS Style**. Water 2019,

11, 951. Disponível em: <https://www.mdpi.com/2073-4441/11/5/951/htm#B21-water-11-00951>. Acesso em 25 jun. 2020.

O'LEARY, Nick. **Arduino PubSubClient - MQTT Client Library Encyclopedia**. Disponível em: <https://www.hivemq.com/blog/mqtt-client-library-encyclopedia-arduino-pubsubclient/>. Acesso em 26 jun. 2020.

ROUSE, Margaret. **Internet of things (IoT)**. Disponível em: <https://internetofthingsagenda.techtarget.com/definition/Internet-of-Things-IoT>. Acesso em 27 jun. 2020.

GOLDENBERG, Barton. **The Internet of Things**. Disponível em: <https://ismguide.com/the-internet-of-things>. Acesso em 27 jun. 2020.

MQTT. **Frequently Asked Questions**. Disponível em: <http://mqtt.org/faq>. Acesso em 27 jun. 2020.

LIGHT, Roger. **MQTT man page**. Disponível em: <https://mosquitto.org/man/mqtt-7.html>. Acesso em 28 jun. 2020.

SCIENCEDAILY. **Speed of sound**. Disponível em: <https://www.sciencedaily.com/terms/speed_of_sound.htm>. Acesso em 29 jun. 2020.

PULTAR, Edward. **How To Monitor Water Levels with Flood Warning Systems and Industrial IoT Sensors**. Disponível em: <https://www.iotforall.com/intro-iot-water-level-sensors>. Acesso em 29 jun. 2020.

TECNOLOGIA, CCM. **Entenda o que é IoT e qual a sua relação com cloud computing**. Disponível em: <https://blog.ccmtecnologia.com.br/post/entenda-o-que-e-iot-e-qual-a-sua-relacao-com-cloud-computing>. Acesso em 30 jun. 2020.

APÊNCIDE A – Código do Microcontrolador Arduino

```
/************************************************************

   Código para leitura de sensor de ultrassom + temperatura e umidade

   Envio por 2G/GPRS para servidor MQTT(Mosquitto)

   Parceria entre UFSCar e Prefeitura Municipal de São Carlos ♥

   Equipe UFSCAR: Prof. Orientador Rafael Vidal Aroca e aluna
   Luana Menezes (Trabalho de Conclusão de Curso em Engenharia de Computação)

   Equipe Prefeitura: Fineias e Evandro


 ************************************************************

   Dependencias do projeto:
    - TinyGSM: https://github.com/vshymanskyy/TinyGSM

    - PubSubClient: https://github.com/knolleary/pubsubclient/releases/tag/v2.8
        - OBS: MQTT_MAX_PACKET_SIZE foi alterado manualmente de 256 para 512 no a
rquivo PubSubClient.h

    - DHT sensor library: https://github.com/adafruit/DHT-sensor-library
        - Sub-dependencia: https://github.com/adafruit/Adafruit_Sensor

    - LinkedList: http://github.com/ivanseidel/LinkedList
        - OBS: Exclua tests.cpp caso tenha, pois há incompatibilidade
        em outras dependencias neste arquivo de testes da biblioteca.

    - Arduino Json: https://github.com/bblanchon/ArduinoJson

   Material utilizado:
    ARDUINO MEGA 2560
    GSM MODEM SIM800L
    SENSOR ULTRASSOM JSN-SR04T-2.0
    DHT11
    REGULADORES DE TENSAO

 ************************************************************/

//**********  ARDUINO JSON  **********
#include <ArduinoJson.h>

//********** WATCHDOG **********
//#include <avr/wdt.h>

//********** TINYGSM **********
#define TINY_GSM_MODEM_SIM800
// Serial para debug
#define SerialMon Serial
// Serial para envio de comandos AT para o módulo
#define SerialAT Serial1
#include <TinyGsmClient.h>

TinyGsm modemGSM(SerialAT);
//TinyGsmClientSecure gsmClient(modemGSM);
TinyGsmClient gsmClient(modemGSM);

//  ---  CONFIGURACOES DE APN   ---
```

```
/*
  const char apn[] = "timbrasil.br";
  const char gprsUser[] = "tim";
  const char gprsPass[] = "tim";*/

const char apn[] = "zap.vivo.com.br";
const char gprsUser[] = "vivo";
const char gprsPass[] = "vivo";

//**********  MQTT **********
#include <PubSubClient.h>

const char* mqtt_server = "167.99.156.110";
const char* mqtt_server_user = "mqtt_root";
const char* mqtt_server_pass = "senha";
const uint32_t mqtt_server_port = 1883;
//const uint32_t mqtt_server_port = 8883;
const char* mqtt_estacao_ID = "estacao01";
const char* mqtt_topico_estacao = "saocarlos/rioX/estacao01";

PubSubClient mqttClient(mqtt_server, mqtt_server_port, gsmClient);
unsigned long tempo_ultimo_envio = 0;
#define INTERVALO_ENVIO 60000

//**********  DHT11 **********
#define DHTPIN 8
#define DHTTYPE DHT11
#include "DHT.h"
DHT dht(DHTPIN, DHTTYPE);
float umidade = 0;
float temperatura = 0;

//**********  ULTRASSOM  **********
#define ULTRA_TRIG_PIN 10
#define ULTRA_ECHO_PIN 11
#define ULTRA_NUM_MEDICOES 20
#define ULTRA_MIN_DISTANCIA 20
#define ULTRA_MAX_DISTANCIA 600
int nivel_agua = 0;

      //********** LOCALIZACAO ESTACAO **********
      #define longitude -47.88067758;
      #define latitude -21.98354135;
      #define altura_sensor 600

      //**********  ESTATISTICA PARA RETIRADA DE OUTLIERS  **********
      #include <LinkedList.h>

      int cmpListaAsc(int &a, int &b) {
        if (a < b) return -1;
        else if (a > b) return 1;
        else return 0;
      }

      void printLista(LinkedList<int> *Lista) {
        int tam_lista = Lista->size();
        SerialMon.print("{ ");
        for (int i = 0; i < tam_lista; i++) {
          int valor = Lista->get(i);
          SerialMon.print(valor);
          SerialMon.print(", ");
        }
        SerialMon.println("}");
      }

```

```
125.       float medianaLista(LinkedList<int> *Lista) {
126.         float mediana;
127.         int tam_lista = Lista->size();
128.         LinkedList<int> lista_aux;
129.
130.         for (int i = 0; i < tam_lista; i++) {
131.           lista_aux.add(Lista->get(i));
132.         }
133.         lista_aux.sort(cmpListaAsc);
134.
135.         if (tam_lista == 0)
136.           return 0;
137.         if (tam_lista % 2 == 0)
138.           mediana = (lista_aux.get(tam_lista / 2) + lista_aux.get(tam_lista / 2
    - 1)) / 2;
139.         else
140.           mediana = lista_aux.get((tam_lista - 1) / 2);
141.
142.         return mediana;
143.       }
144.
145.       //Median Average Deviation
146.       float MAD(LinkedList<int> *Lista) {
147.         float mad, mediana;
148.         int tam_lista = Lista->size();
149.         LinkedList<int> lista_aux;
150.
151.         mediana = medianaLista(Lista);
152.         for (int i = 0; i < tam_lista; i++) {
153.           lista_aux.add(fabsf(Lista->get(i) - mediana));
154.         }
155.         mad = medianaLista(&lista_aux);
156.         return mad;
157.       }
158.
159.       //Z Score Modificado
160.       int zScoreMod_MedianaSemOutliers(LinkedList<int> *Lista, float beta = 1.0)
    {
161.         int mediana, mediana_sem_outliers;
162.         int tam_lista = Lista->size();
163.         LinkedList<float> lista_dif_mediana, lista_mi;
164.         LinkedList<int> lista_sem_outliers;
165.         //LinkedList<int> lista_outliers;
166.
167.         mediana = medianaLista(Lista);
168.         for (int i = 0; i < tam_lista; i++) {
169.           lista_dif_mediana.add(fabsf(Lista->get(i) - mediana));
170.         }
171.
172.         int mad = MAD(Lista);
173.         if (mad == 0) {
174.           mad = 1;
175.         }
176.
177.         for (int i = 0; i < tam_lista; i++) {
178.           lista_mi.add((lista_dif_mediana.get(i) * 0.6745) / mad);
179.         }
180.         int outlier; float mi;
181.         for (int i = 0; i < tam_lista; i++) {
182.           mi = lista_mi.get(i);
183.
184.           if (mi >= beta) {
185.             //outlier = Lista->get(i);
186.             //lista_outliers.add(outlier);
187.           } else {
```

```
188.             lista_sem_outliers.add(Lista->get(i));
189.           }
190.         }
191.         mediana_sem_outliers = medianaLista(&lista_sem_outliers);
192.         return mediana_sem_outliers;
193.       }
194.
195.       //  ---   CONFIGURAR O MODEM GSM ---
196.       boolean iniciarModemGSM() {
197.
198.         SerialMon.println("Iniciando modem...");
199.         modemGSM.restart();
200.         delay(10000);
201.         String modem_info = modemGSM.getModemInfo();
202.         SerialMon.print("Modem Info: ");
203.         SerialMon.println(modem_info);
204.
205.         int qualidade_sinal = 0;
206.         SerialMon.println("Verificando qualidade do sinal da rede...");
207.         for (int i = 0; i < 5; i++) {
208.           qualidade_sinal = modemGSM.getSignalQuality();
209.           if (qualidade_sinal == 0) {
210.             SerialMon.print("Fraco: ");
211.           } else if (qualidade_sinal == 99) {
212.             SerialMon.print("Desconhecido: ");
213.           } else {
214.             SerialMon.print("Sinal: ");
215.           }
216.           SerialMon.print(qualidade_sinal);
217.           SerialMon.print(" | ");
218.           delay(100);
219.         }
220.         SerialMon.println();
221.         SerialMon.println();
222.         SerialMon.print("Procurando rede...");
223.         if (!modemGSM.waitForNetwork()) {
224.           SerialMon.println(" falha");
225.           delay(1000);
226.           return false;
227.         }
228.         SerialMon.println(" sucesso!");
229.
230.         if (modemGSM.isNetworkConnected()) {
231.           SerialMon.println("Rede conectada");
232.         }
233.
234.         SerialMon.print("Conectando com ");
235.         SerialMon.print(apn);
236.         SerialMon.print("...");
237.         if (!modemGSM.gprsConnect(apn, gprsUser, gprsPass)) {
238.           SerialMon.println(" falha");
239.           delay(1000);
240.           return false;
241.         }
242.         SerialMon.println(" sucesso!");
243.
244.         if (modemGSM.isGprsConnected()) {
245.           SerialMon.println("GPRS conectado");
246.           return true;
247.         }
248.       }
249.
250.       //  ---   (RE)CONECTAR O CLIENTE MQTT ---
251.       boolean conectarMQTT() {
252.         SerialMon.print("Conectando com MQTT Server ");
```

```
253.         SerialMon.print(mqtt_server);
254.         SerialMon.print("...");
255.
256.         if (mqttClient.connect(mqtt_estacao_ID, mqtt_server_user , mqtt_server_p
    ass)) {
257.           SerialMon.println(" sucesso!");
258.           SerialMon.println("MQTT conectado");
259.           return mqttClient.connected();
260.         } else {
261.           SerialMon.println(" falha");
262.           SerialMon.print("MQTT nao conectado. ");
263.           //tabela de erros disponivel em: https://pubsubclient.knolleary.net/ap
    i#state
264.           SerialMon.print("Erro = ");
265.           SerialMon.println(mqttClient.state());
266.           return false;
267.         }
268.       }
269.
270.       //  --
    -   CALLBACK DO CLIENTE MQTT PARA SUBSCRIBE (NÃO USADO NO CODIGO ATUAL)   ---
271.       void mqttCallback(char* topico, byte* payload, unsigned int len) {
272.         SerialMon.print("Mensagem recebida [");
273.         SerialMon.print(topico);
274.         SerialMon.print("]: ");
275.         SerialMon.write(payload, len);
276.         SerialMon.println();
277.       }
278.
279.       //  ---  PUBLICACAO MQTT TEMPERATURA, UMIDADE  E DISTANCIA ---
280.       boolean publicarTempUmiNivelNoTopico() {
281.         StaticJsonDocument <256> RAMbuffer;
282.         JsonObject JSONencoder = RAMbuffer.to<JsonObject>();
283.
284.         JSONencoder["estacao_ID"] = mqtt_estacao_ID ;
285.         JSONencoder["distancia"] = nivel_agua;
286.         JSONencoder["umidade"] = umidade;
287.         JSONencoder["temperatura"] = temperatura;
288.         JSONencoder["topico"] = mqtt_topico_estacao;
289.         JSONencoder["altura"] = altura_sensor;
290.
291.         JsonObject coord = JSONencoder.createNestedObject("coord");
292.         coord["lon"] = longitude;
293.         coord["lat"] = latitude;
294.
295.         char JSONmessageBuffer[256];
296.         //apenas para debug - o envio será do Json minificado
297.         serializeJsonPretty(RAMbuffer, JSONmessageBuffer);
298.         SerialMon.print(JSONmessageBuffer);
299.         size_t n = serializeJson(RAMbuffer, JSONmessageBuffer);
300.
301.         if (mqttClient.publish(mqtt_topico_estacao, JSONmessageBuffer, n)) {
302.           return true;
303.         }
304.         else {
305.           return false;
306.         }
307.       }
308.
309.       // --  METODOS DE LEITURA DOS SENSORES  ---
310.       boolean medirTemperaturaUmidade() {
311.         float h = dht.readHumidity();
312.         float t = dht.readTemperature(); // leitura da temperatura em graus C
313.
314.         if (isnan(h) || isnan(t)) {
```

```
315.           SerialMon.println(("Falha do sensor DHT"));
316.           return false;
317.         }
318.
319.         umidade = h;
320.         temperatura = t;
321.         SerialMon.println();
322.         SerialMon.print("Umidade: ");
323.         SerialMon.print(umidade);
324.         SerialMon.print("%  Temperatura: ");
325.         SerialMon.print(temperatura);
326.         SerialMon.println("°C ");
327.
328.         return true;
329.         //delay(250);
330.       }
331.
332.       long microsegundosParaCentimetros(long microsegundos) {
333.         return microsegundos / 29 / 2;
334.       }
335.
336.       boolean medirNivelAgua() {
337.         LinkedList<int> lista_ultrassom;
338.
339.         unsigned long duracao, cm, soma_cm = 0;
340.         float cm_real = 0;
341.         int c = 0;
342.
343.         for (int i = 0; i < ULTRA_NUM_MEDICOES; i++) {
344.           digitalWrite(ULTRA_TRIG_PIN, LOW);
345.           delayMicroseconds(2);
346.           digitalWrite(ULTRA_TRIG_PIN, HIGH);
347.           delayMicroseconds(15);
348.           digitalWrite(ULTRA_TRIG_PIN, LOW);
349.
350.           duracao = pulseIn(ULTRA_ECHO_PIN, HIGH, 10000000);
351.
352.           cm_real = microsegundosParaCentimetros(duracao);
353.           cm = round(cm_real);
354.
355.           if ((cm >= ULTRA_MIN_DISTANCIA) && (cm <= ULTRA_MAX_DISTANCIA)) {
356.             lista_ultrassom.add(cm);
357.           }
358.
359.           SerialMon.print(cm);
360.           SerialMon.print("cm");
361.           SerialMon.print(" | ");
362.           int pula_linha = i + 1;
363.           if (pula_linha % 10 == 0 && (pula_linha != ULTRA_NUM_MEDICOES)) Serial
    Mon.println();
364.
365.           // leva 38ms para leitura de distancia maxima
366.           delay(40);
367.         }
368.
369.         if (lista_ultrassom.size() == 0) {
370.           SerialMon.println("Falha no sensor de ultrassom");
371.           return false;
372.         }
373.         cm = zScoreMod_MedianaSemOutliers(&lista_ultrassom, 0.7);
374.         nivel_agua = cm;
375.         SerialMon.println();
376.         SerialMon.print("Mediana sem Outliers: ");
377.         SerialMon.print(nivel_agua);
378.         SerialMon.println("cm");
```

```
379.
380.         return true;
381.       }
382.
383.       //------------------------------------------------------------setup--------
    ----------------------
384.       void setup() {
385.         SerialMon.begin(115200);
386.         while (!SerialMon);
387.
388.         pinMode(ULTRA_TRIG_PIN, OUTPUT);
389.         pinMode(ULTRA_ECHO_PIN, INPUT);
390.
391.         SerialMon.println("");
392.         SerialMon.println("Espere...");
393.
394.         SerialAT.begin(9600);
395.         while (!SerialAT);
396.
397.         while (!iniciarModemGSM()) {
398.           SerialMon.println("Falha Setup GSM...Tentando Novamente...");
399.         }
400.
401.         conectarMQTT();
402.
403.         //Setando funcao callback para recebimento de mensagens ao dar subscribe

404.         mqttClient.setCallback(mqttCallback);
405.
406.         //Inicializando o sensor DHT11
407.         dht.begin();
408.       }
409.       //------------------------------------------------------------------loop-----
    ----------------------
410.       void loop() {
411.
412.         if (!modemGSM.isGprsConnected()) {
413.           SerialMon.println("Reconectando GPRS & MQTT...");
414.           while (!iniciarModemGSM()) {
415.             SerialMon.println("Falha Reconexao GSM...Tentando Novamente...");
416.           }
417.         }
418.
419.         if (!mqttClient.connected()) {
420.           SerialMon.println();
421.           SerialMon.println("Reconectando MQTT...");
422.           conectarMQTT();
423.         } else {
424.           unsigned long tempo_agora = millis();
425.           if (tempo_agora - tempo_ultimo_envio > INTERVALO_ENVIO) {
426.             if (medirTemperaturaUmidade() && medirNivelAgua()) {
427.               SerialMon.print("Enviando pro MQTT Server...");
428.
429.               if (publicarTempUmiNivelNoTopico()) {
430.                 SerialMon.println("publicado com sucesso!");
431.               } else {
432.                 SerialMon.println("falha ao enviar!");
433.               }
434.
435.               tempo_ultimo_envio = tempo_agora;
436.               SerialMon.print("Aguardando o intervalo em milisegundos: ");
437.               SerialMon.println(INTERVALO_ENVIO);
438.             } else {
439.               //SerialMon.print("Erro de leitura nos sensores...");
440.             }
```

```
441.           }
442.         }
443.         mqttClient.loop();
444.       }
```

APÊNCIDE B – Código do Cliente Python (mqtt_to_bd.py)

```python
import paho.mqtt.client as paho
import json
import mysql.connector as db
from mysql.connector import errorcode

# BD configs
db_host = "db-mysql-nyc1-94857-do-user-7555041-0.a.db.ondigitalocean.com"
db_port = "25060"
db_user = "doadmin"
db_pass = "senha"
db_database = "tcc_db"

# Mqtt configs
mqtt_server = "167.99.156.110"
mqtt_port = 1883
mqtt_topico = "saocarlos/#"
mqtt_user = "mqtt_root"
mqtt_pass = "senha"

connection = None

def connectionToDb():
    global connection
    try:
        if not connection:
            connection = db.connect(
                host=db_host,
                port=db_port,
                user=db_user,
                password=db_pass,
                database=db_database
            )
            return connection
    except db.Error as error:
        if error.errno == errorcode.ER_BAD_DB_ERROR:
            print("O banco de dados não existe")
        elif error.errno == errorcode.ER_ACCESS_DENIED_ERROR:
            print("O usuario e/ou senha incorretos")
        else:
            print(error)

def insertIndoDb(id_estacao, lon, lat, dist, temp, um, top, altura):
    try:
        mycursor = connection.cursor()
        sql = "INSERT INTO tcc_db.sensor_historico (id_client_estacao, longitude, latitude, distancia, temperatura, umidade, topico, data, altura_sensor) VALUES (%s,%s, %s, %s, %s, %s, %s, now(), %s);"
        val = (id_estacao, lon, lat, dist, temp, um, top, altura)
        mycursor.execute(sql, val)
        connection.commit()
        print("Inserido com sucesso" )
        #connection.close()
    except Exception as e:
        print(e)

def on_log(client, userdata, level, buf):
    print("log: "+buf)

def on_connect(client, userdata, flags, rc):
    if rc == 0:
```

```
        print("MQTT Conectado, codigo="+str(rc))
    else:
        print("MQTT falhou, codigo="+str(rc))

def on_message(client, userdata, msg):
    print(msg.payload)
    payload = json.loads(msg.payload)
    id_estacao = payload["estacao_ID"]
    lon = payload["coord"]["lon"]
    lat = payload["coord"]["lat"]
    dist = payload["distancia"]
    temp = payload["temperatura"]
    um = payload["umidade"]
    top = payload["topico"]
    altura = payload["altura"]
    insertIndoDb(id_estacao, lon, lat, dist, temp, um, top, altura)

connection = connectionToDb()

mqtt_client = paho.Client("data-to-db")
mqtt_client.on_connect = on_connect
mqtt_client.on_message = on_message
mqtt_client.on_log = on_log

print("Conectando com o servidor MQTT...")
mqtt_client.username_pw_set(mqtt_user, mqtt_pass)
mqtt_client.connect(mqtt_server, mqtt_port)
mqtt_client.subscribe(mqtt_topico)

mqtt_client.loop_forever()
```

APÊNCIDE C – Código do Banco de Dados

```sql
CREATE DATABASE  IF NOT EXISTS "tcc_db";
USE `tcc_db`;
DROP TABLE IF EXISTS `sensor_historico`;
CREATE TABLE `sensor_historico` (
  `id_mensagem` int NOT NULL AUTO_INCREMENT,
  `id_client_estacao` varchar(50) NOT NULL,
  `longitude` float NOT NULL,
  `latitude` float NOT NULL,
  `distancia` smallint NOT NULL,
  `temperatura` smallint NOT NULL,
  `umidade` smallint NOT NULL,
  `topico` varchar(50) NOT NULL,
  `data` timestamp NULL DEFAULT CURRENT_TIMESTAMP,
  `altura_sensor` smallint NOT NULL,
  PRIMARY KEY (`id_mensagem`)
)
```