UNIVERSIDADE FEDERAL DO ABC

RICARDO NOGUEIRA AMBROSIO

DESENVOLVIMENTO DE APLICAÇÃO PARA IoT PARA SENSORIAMENTO DE NÍVEL VOLUMÉTRICO E DETECÇÃO DE ENCHENTES

SANTO ANDRÉ - SP
2017

RICARDO NOGUEIRA AMBROSIO

DESENVOLVIMENTO DE APLICAÇÃO PARA IoT PARA SENSORIAMENTO DE NÍVEL VOLUMÉTRICO E DETECÇÃO DE ENCHENTES

Parte III do Trabalho de Graduação apresentado à Universidade Federal do ABC, como requisito para obtenção do diploma no curso de graduação em Engenharia de Informação.
Orientador: Prof. Dr. João Henrique Kleinschmidt

SANTO ANDRÉ - SP
2017

RICARDO NOGUEIRA AMBROSIO

DESENVOLVIMENTO DE APLICAÇÃO PARA IoT PARA SENSORIAMENTO DE NÍVEL VOLUMÉTRICO E DETECÇÃO DE ENCHENTES

Parte III do Trabalho de Graduação apresentado à Universidade Federal do ABC, como requisito para obtenção do diploma no curso de graduação em Engenharia de Informação, avaliado pela comissão formada por:

_____
Prof. Dr. João Henrique Kleinschmidt
UFABC

_____
Prof. _____
Instituição _____

SANTO ANDRÉ – SP
2017

RESUMO

Casos de enchentes estão presentes na mídia e não são raros. O problema atual dos rios se dá pelo excesso de detritos acumulados que somado com chuvas volumosas provocam o transbordamento do rio. Caso isso aconteça as residências e estabelecimentos locais ficam à mercê da percepção no momento, porém, havendo um sistema para detecção de transbordamento, seria possível ter informações em tempo real sobre o estado volumétrico do rio, bem como informações sobre a taxa de elevação do nível. Às pessoas locais seriam emitidos alertas caso o nível atingisse um limiar crítico, indicando a necessidade de evacuação.

Foi realizada a construção de dois sistemas de medições para dois reservatórios distintos. Um módulo central chamado *coordinator* que possui conectividade com a internet e, portanto, o envio dos dados, e um módulo unicamente de medições que se chama *router*. Esse módulo envia as informações para o módulo central, que efetua os cálculos de criticidade e atualiza na plataforma *ThingSpeak*.

Durante o projeto percebeu-se a viabilidade de implementação em um ambiente real para futuros testes, pois no ambiente controlado obteve-se resultados significativos e condizentes com os propostos. Conforme o nível volumétrico aumenta, a valor da criticidade aumenta junto, seguindo a tabela proposta de criticidade.

Com um ajuste na programação, para ambientes maiores, acredita-se na viabilidade do projeto para realizar medições num ambiente real.

Palavras-chave: *internet* das coisas, cidades inteligentes, detecção de enchente, arduino, automação, ESP8266, shield ethernet, XBee, sensor de nível volumétrico, *thingspeak*.

SUMÁRIO

1 Introdução

A *internet* está a cada dia mais presente na rotina das pessoas e sendo utilizada para diversas finalidades. Segundo o Instituto Brasileiro de Geografia e Estatística (IBGE) em 2013 metade dos brasileiros obtiveram acesso à *internet* e esses valores tendem a crescer a cada ano que se passa à medida que planos de *internet* (tanto residenciais quanto móveis) se tornem economicamente mais atrativo e aumentem sua capacidade de velocidade ao cliente final [1].

Uma mostra do potencial de crescimento global da *internet* se dá através da necessidade do desenvolvimento de um novo protocolo para a *internet*, o IPv6, para gradualmente substituir o utilizado atualmente (IPv4) que suporta cerca de 4 bilhões de endereços IP distintos (2^{32}), número menor do que a população mundial, enquanto que o novo protocolo IPv6 tem cerca de $3{,}4 \times 10^{38}$ endereços distintos (2^{128}) [2]. Com a modernização do protocolo IP, começaria a perder força o NAT (tradução de endereço de redes), pois na versão mais nova têm-se muito mais endereços disponíveis do que pessoas e dispositivos juntos no mundo, assim, viabilizando a utilização de dispositivos com endereço IP único, fortalecendo a ideia de *Internet of Things* (IoT), internet das coisas, em português.

Visando a interligação do cotidiano com a tecnologia, começa a surgir cada dia que passa matérias em jornais, soluções comerciais e projetos (tanto de pesquisa quanto de execução), tornando o foco em possíveis soluções úteis e que agregam valor ao usuário. Com isso em mente, começa-se a pensar em situações do cotidiano sob a perspectiva da internet das coisas.

O exemplo que será amplamente tratado nesse projeto é relativo às enchentes. Em 2014 foi lançado um relatório do IBGE em que foi levantado que 1543 dos 5570 municípios brasileiros foram atingidos por enchentes ou inundações graduais nos últimos 5 anos (período entre 2008 e 2013). Desses 1543 municípios houve um total de mais de 655 mil edificações atingidas em 1406 municípios e mais de um milhão e quatrocentos mil pessoas ficaram desalojadas ou desabrigadas em 1306 municípios [3].

Com esses dados do IBGE pode ser visto o impacto positivo que teria um sistema de detecção de enchentes para poder acompanhar o nível do rio, e na eventualidade do mesmo correr risco de transbordar, poderia ser emitido um alerta às comunidades próximas.

Um exemplo vigente pode ser analisado na cidade de Santo André, São Paulo, onde em conjunto com a SEMASA (órgão de saneamento ambiental) criou-se uma página para que seja possível monitorar os principais córregos da cidade através de câmeras posicionadas em locais estratégicos dos rios, com imagens atualizadas a cada trinta segundos [4].

Foi escolhido esse tema de projeto como tentativa de prover retorno à comunidade, para estudar a viabilidade de monitorar os rios a fim de detectar um possível alagamento em épocas de chuvas torrenciais (ou pelo menos nos principais rios).

O presente trabalho tem por objetivo criar um protótipo em escala reduzida e modular de um conjunto de sensores para detecção de enchentes. Espera-se que a captação de dados seja realizada em tempo real, para que seja possível determinar a velocidade e criticidade com que o nível volumétrico de um reservatório varia. Ao término dos testes e projeto, será determinada a viabilidade de implementação dessa solução em ambiente real.

No capítulo 2 será descrito internet das coisas e cidades inteligentes. No capítulo 3 será realizado um breve resumo das tecnologias utilizadas no projeto, que englobam Arduino, ethernet, XBee, a plataforma *ThingSpeak* e o sensor ultrassônico. O capítulo 4 envolve a metodologia utilizada no projeto, com descrições dos materiais utilizados e metodologia utilizada. No capítulo 5 inicia-se a apresentação de resultados e discussões. A conclusão é apresentada no capítulo 6 e nos anexos A e B são dispostos os códigos completos utilizados no projeto.

2 Internet das Coisas e Cidades Inteligentes

Dado o amplo acesso à *internet*, e à evolução tecnológica (ambos *hardware* e *software*) abriu-se a oportunidade de crescimento de um campo denominado *Internet* das Coisas (em inglês, *Internet of Things*), termo cunhado em 1999, através da utilização da eletrônica em objetos e dispositivos do cotidiano de uma pessoa tornando-os sistemas (ou objetos) inteligentes [5]. Os objetos podem variar desde celulares, videogames, *GPS*, sensores diversos, iluminação (pública e/ou residencial ou semáforos), câmeras de trânsito e até mesmo utensílios de cozinha como micro-ondas, geladeiras e cafeteiras. Essencialmente, objetos e dispositivos que possuam circuito e algum tipo de conexão.

Com o fortalecimento da *internet* das coisas a *internet* como antes conhecida por interligar dispositivos de usuários de uma ponta a outra (herança desde a criação da ARPANET em 1969), está sofrendo uma mudança de ponto de vista, e passando a ser vista como uma *internet* que conecta objetos físicos a outros objetos e/ou pessoas a fim de prover algum tipo de funcionalidade [6]. O conceito de *internet* das coisas se baseia em três conceitos de que os objetos inteligentes devam conseguir: identificar-se, comunicar-se e interagir entre si [5].

Atualmente, estima-se que existam cerca de 15 bilhões de dispositivos e objetos conectados à rede e que até o ano de 2020 essa quantidade passe para 50 bilhões, segundo o estudo feito pela companhia Cisco, multinacional americana de tecnologia que atua com soluções para redes e comunicações [7]. Outra estimativa relevante a ser citada é a do Conselho Nacional de Inteligência dos Estados Unidos da América (*NIC*) que prevê que até o ano de 2025 nós conectados à *internet* residirão nos itens do dia a dia como pacotes de comidas, móveis, documentos de papel, entre outros itens cotidianos [8]. Para isso ser viável, alguns desafios a frente deverão ser vencidos, como a miniaturização desses dispositivos, a diminuição do consumo energético e computacional do mesmo e a parte de segurança da informação trafegada, bem como da rede com um todo, respeitando os pilares da segurança da informação (confiabilidade, integridade e disponibilidade).

Uma aplicação em potencial é no domínio da saúde, onde um determinado paciente possuiria uma identificação, possivelmente um *chip* de identificação de rádio frequência (*RFID*), e neste *chip* conteria todo os dados e histórico médico do paciente, podendo então minimizar danos referentes à negligência de procedimentos, como

dosagem de medicamento, ou aplicação de medicamentos aos quais o paciente é alérgico. Outro exemplo a ser citado é a possibilidade de que uma pessoa diabética poderia realizar a checagem da quantidade de açúcar no sangue em sua própria casa, e esse sensor enviaria os dados automaticamente para o seu médico de preferência, que poderia ter um controle otimizado em relação a condição do paciente, podendo agendar uma consulta de acordo com o que julgar necessário [9].

Algumas aplicações que estão mais comuns são no domínio de ambientes inteligentes, como por exemplo comodidade e segurança residencial com sensores e atuadores distribuídos pela casa. Desse modo seria possível centralizar informações sobre luzes acesas, oferecendo a possibilidade de desliga-las remotamente, portas e/ou janelas abertas, temperatura do ambiente (e se ligado a uma estação HVAC, controlar o aquecimento, ventilação e ar condicionado) ou tomadas inteligentes com informação de consumo de energia [9].

A expansão e modernização da tecnologia no campo da automação e *internet* das coisas, fornece um ambiente propício para o crescimento conjunto da ideia de cidade inteligente, cuja definição é dada abaixo:

> O uso de tecnologias de computação inteligente para fazer componentes críticos de infraestrutura e serviços de uma cidade – que incluem administração, educação, saúde, segurança pública, imóveis, transporte e utilidades – mais inteligente, interconectados e eficientes. (WASHBURN et. al., 2010, p. 2, tradução nossa)

A computação inteligente, mencionada acima, consiste na tomada de decisões inteligentes, usando como base as informações provenientes dos objetos conectados na rede. Consiste da junção de três grandes fatores: institucionais (como governança, política e diretrizes), humanos (como educação e criatividade) e tecnológicos (como tecnologias inteligentes, tecnologias móveis e redes digitais) [10]. Enquanto *internet* das coisas tem uma aplicação direta mais técnica, o conceito de cidades inteligentes acaba sendo mais macro e leva em conta fatores gerenciais e políticos também, tornando a *internet* das coisas um membro dentro do fator tecnológico de cidades inteligentes [11].

O aumento de pesquisas e artigos nessas áreas pode trazer um benefício à população. Desde 2004 até 2015, o aumento de artigos publicados nas áreas em questão foi notável.

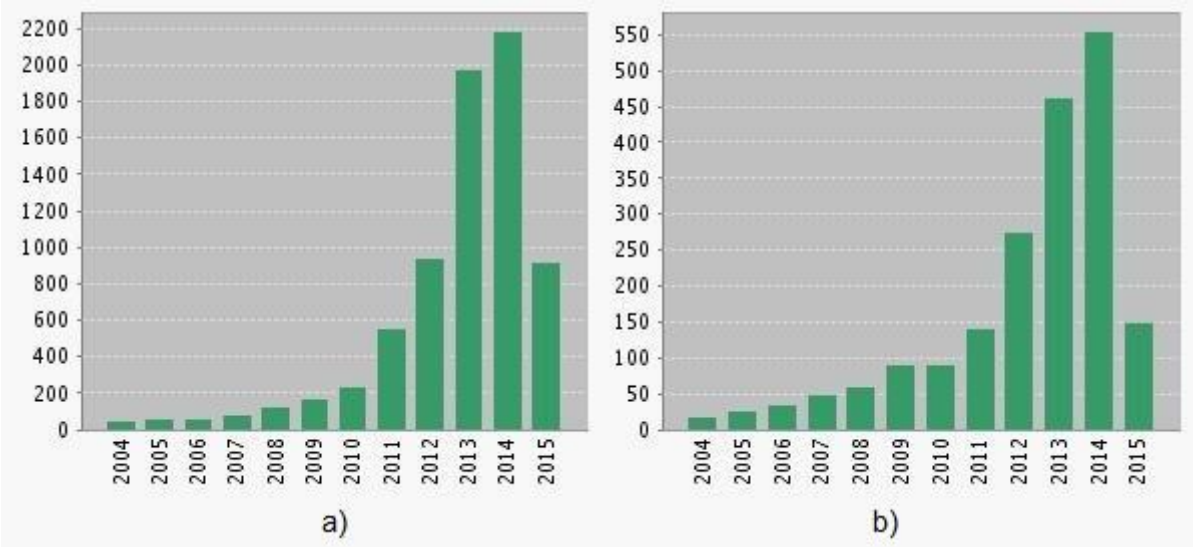


Figura 1: A figura a) mostra o aumento de artigos desde o ano de 2004 até o meio do ano de 2015 a respeito do tópico a) "*internet* das coisas", e sobre b) "cidade inteligente", utilizando como base de dados os *Web of Science®.*

Chuvas constantes acabam deixando o rio em um estado saturado, que não consegue dar vazão necessária em seu volume de água, o que pode acontecer devido ao solo encharcar-se e não conseguir absorver o excesso, ou devido ao acúmulo de detritos carregados durante temporais que possam entupir a saída do rio, fazendo com que o equilíbrio de entrada de água e saída seja prejudicado. Residências e outros estabelecimentos que ficam próximos aos rios acabam ficando sujeitos a uma enchente nesse caso (o mesmo ocorre para chuvas torrenciais esporádicas, que mesmo sem a saturação do rio, não consegue ocorrer vazão devido ao alto volume de água que chega de uma só vez).

Com um sistema de detecção de enchentes seria possível monitorar não apenas qualitativamente, como no caso do serviço disponível no site da SEMASA, como também quantitativamente, visto que seria possível retirar dados do nível atual do rio.

3 Tecnologias Utilizadas

A ideia base da *internet* das coisas consiste em tornar sistemas (ou objetos) inteligentes, ou seja, interconectados em uma rede e que sejam capazes de desempenhar funções ou realizar tomadas de decisões com base em uma programação [5]. Para que isso seja possível, vê-se necessário a utilização das seguintes tecnologias. O arduino, para a configuração principal do projeto, contendo toda lógica e tomada de decisão, o *shield* ethernet para conexão com a internet, e prover o meio necessário para o envio de dados à plataforma. O XBee para a comunicação inter-arduino, de modo a centralizar a conexão com a *internet* em apenas um arduino. Os sensores serão os responsáveis pelas aquisições de dados e a plataforma *ThingSpeak* será utilizada para a exibição dos mesmos.

3.1 Arduino

Existe um grande número de placas diferentes, visto que o Arduino é uma placa baseada em microcontroladores e *software* e *hardware open-source*. O Arduino Uno é uma placa microcontroladora baseada no chip ATmega328P, da fabricante Atmel. As 14 portas de dados digitais tanto de *input* como de *output*, e 6 portas analógicas. Possui uma tensão de operação entre 5 volts com entrada recomenda entre 7 e 12 volts, sendo compatível com a grande maioria das tensões máximas dos módulos utilizados no mercado. Por possuir uma comunidade vasta de utilização é possível obter suporte online para diversos problemas, e bibliotecas prontas para utilização de diversos módulos [12].

Figura 2: Representação de um Arduino Uno R3.

3.2 ESP8266, Shield Ethernet e Zigbee

Para os módulos de comunicação temos as seguintes tecnologias ESP8266 (IEEE 802.11), Shield Ethernet W5100 (IEEE 802.3) e o Zigbee (IEEE 802.15.4). O módulo ESP8266 pode funcionar de três modos, como *Access Point*, roteador, e *Access Point* e Roteador, desse modo é possível conectar o dispositivo à uma rede já existente, criar a sua própria, ou conectar a uma rede e criar uma própria, separando a rede de *internet* com a de dados do módulo [13].

Figura 3: Representação do módulo de comunicação Wi-Fi, ESP8266.

O módulo ethernet W5100 tem funcionamento integrado. Feito especificamente para o Arduino, o módulo encaixa diretamente no Arduino e sua biblioteca para funcionamento faz parte do conjunto base de bibliotecas da IDE do Arduino. No módulo já vem equipado com pilha TCP/IP incorporada, desse modo os protocolos convencionais TCP, UDP, IPv4, ICMP, ARP, IGMP e PPPoE são facilmente utilizáveis em diversos projetos [14].

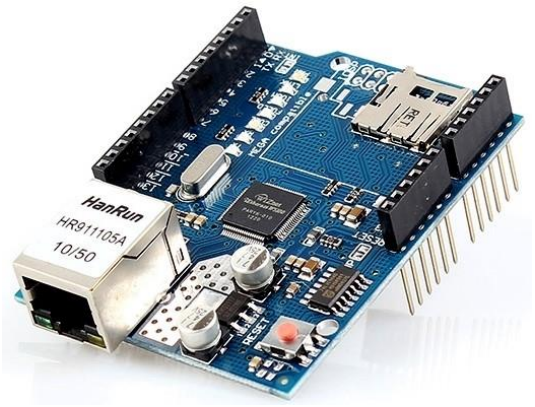


Figura 4: Representação do módulo de ethernet W5100 para o Arduino.

Já o Zigbee funciona com um protocolo diferente do IEEE 802.11 comumente utilizado pelo Wi-Fi, e utiliza-se o protocolo IEEE 802.15.4. Suas camadas são compostas de Aplicação, framework de aplicação, rede e segurança, MAC e física, porém as camadas que são definidas pelo IEEE 802.15.4 são apenas as duas camadas mais baixas (MAC e físico). A especificação do Zigbee adiciona camadas lógicas acima das camadas implementadas pelo IEEE 802.15 [15].

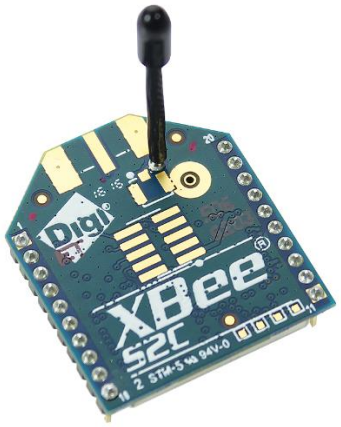


Figura 5: Representação do XBee Series 2.

Similar a protocolos como o Wi-Fi, o Zigbee utiliza canais CSMA/CA (*Carrier Sense Multiple Access with Collision Avoidance*) para tentar evitar colisões e perdas de pacotes durante a comunicação, enquanto Ethernet utiliza CSMA/CD (*Carrier Sense Multiple Access with Collision Detection)* como meio de acesso ao canal.

CSMA/CD é o método de acesso à LAN utilizado pelo protocolo Ethernet. Quando um dispositivo necessita acesso à rede, é realizada a transmissão do *frame*, caso ocorra uma colisão (tenha mais alguém transmitindo ou recebendo dados no momento) o dispositivo aguarda um período aleatório antes de tentar novamente, isso faz com que diminua a tentativa conjunta de dispositivos tentando acessar a rede para envio de informações.

O CSMA/CA, utilizado pelas redes Wi-Fi, por sua vez difere do CD por analisar o meio de propagação ao invés de simplesmente tentar transmitir, e retransmitir caso alguém esteja utilizando o meio. Portando, CSMA/CA evita as colisões por não transmitir informações caso alguém estiver transmitindo.

O pelo fato de ser menor e mais leve que os outros protocolos, consome menos energia também, viabilizando a utilização em dispositivos menores [5]. Uma das funcionalidades do Zigbee é a criação da rede no estilo *Mesh*, que garante uma

interconexão de nós que faz com que o dado seja retransmitido entre pontos intermediários até que atinja o seu destino [17].

O Zigbee conta com um alcance médio de aproximadamente 100 metros em um ambiente interno e aproximadamente 1200 metros em um ambiente externo em situações ideais, porém com uma taxa de transmissão menor em torno de 250 kbps, que garante um menor consumo energético, como mencionando acima, tornando-o ideal para projetos de baixo alcance [17].

A rede Zigbee foi projetada para utilizar um dispositivo como *coordinator*, que pode-se fazer uma analogia com os roteadores de borda de uma ISP. Um segundo e terceiro tipo de dispositivo presente na rede Zigbee chamam-se *router* e *end devices*, que pode-se fazer uma analogia com os roteadores de uma casa que fazem a comunicação entre os computadores e celulares conectados à rede local com a rede de sua ISP de escolha, como visto na imagem abaixo.

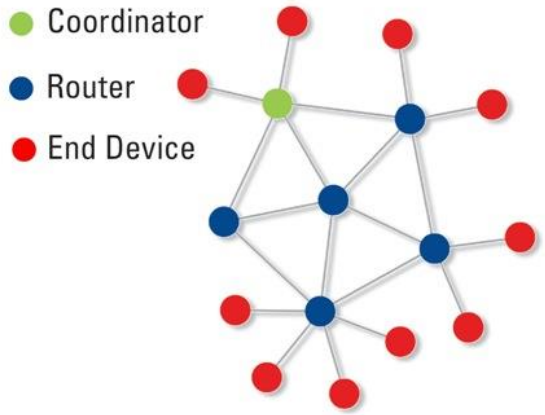


Figura 6: Caracterização de uma rede convencional Zigbee.

3.3 Sensoriamento

Para aquisição de dados foi utilizado o sensor HC-SR04, um sensor ultrassônico. Esse sensor, conta com quatro pinos distintos, VCC e GND, para alimentação de 5 volts e aterramento, e *Trigger* e *Echo*, para emissão do sinal e recebimento do sinal na volta [21].

Figura 7: Representação do sensor ultrassônico HC-SR04.

A cabeça emite automaticamente 8 pulsos de 40kHz após uma entrada de 10 microssegundos de sinal alto no pino *Trigger* conectado ao Arduino, como visto na figura 8 abaixo. Após essa emissão, o *Trigger* é colocado em sinal baixo, e ativado o pino *Echo*, que fica aguardando o retorno do sinal, uma vez que haja esse retorno, é calculado o tempo de navegação (ida e volta) do sinal, normalmente na escala de microssegundos, como visto na figura 9.

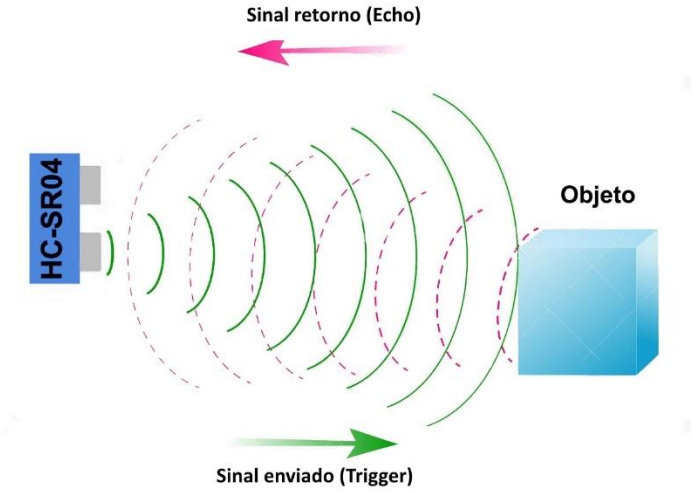


Figura 8: Representação do sinal gerado pelo *trigger* (em verde) e o retorno do sinal pelo *echo* (em rosa).

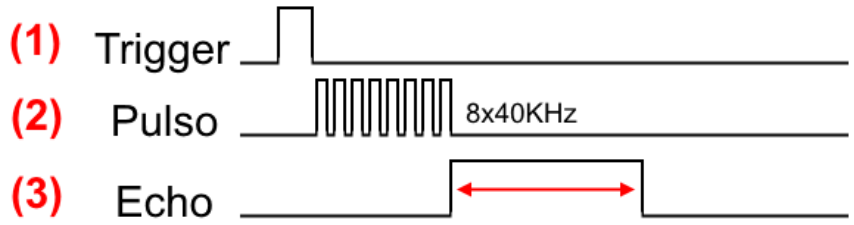


Figura 9: Representação das 3 fases do sensor ultrassônico. Envio do sinal alto ao pino do *trigger* no arduino na fase (1), pulso modulado em 40 kHz na fase (2) e obtenção da duração do *echo* (retorno) na fase (3).

Para calcular a distância, utiliza-se a equação 1 abaixo.

$$Distância = \frac{(Tempo\ ECHO\ em\ Nível\ Alto\ \times Velocidade\ do\ Som\ no\ Ar)}{2} \qquad (1)$$

De acordo com o *datasheet* do sensor, o mesmo possui uma abertura ótima de operação de aproximadamente 15º [21].

Utiliza-se a biblioteca *"NewPing"* para facilitar a aquisição de dados. Onde existe métodos pronto para ativação do *trigger*, obtenção do *echo* e conversão do tempo para distância (em centímetros) com facilidade.

3.4 ThingSpeak

Parte do portfólio de soluções da empresa *Mathworks*, a plataforma ThingSpeak permite que dados sejam agregados, analisados e visualizados na nuvem (remotamente).

Dentre as características presentes da plataforma, destacam-se as abaixo.

- Fácil configuração utilizando protocolos populares de internet das coisas.
- Visualização de dados de sensores em tempo real.
- Agregação de dados de fontes terceiras.
- Utilização de MATLAB para análise dos dados coletados.
- Atuações automáticas com base em dados coletados/analisados e utilização de serviços externos (como por exemplo, *Twitter*).

Para intervalos de atualização menores do que 16 segundos, é necessário a contratação de um plano da plataforma.

Muito utilizado para testes de prova de conceito, uma das vantagens de ter-se a plataforma na nuvem (ainda mais com integração ao MATLAB) é sua visualização rápida, prática e com análise rápida de dados, não sendo necessário dispor de um servidor próprio para realização dos testes, que por vezes, pode dificultar o funcionamento de *softwares* que necessitem poderes computacionais mais elevados, como o MATLAB.

Figura 10: Representação da visualização geral da plataforma.

A plataforma conta com modelos de códigos para importação de dados no MATLAB, bastando clicar em *"MATLAB Analyses"* ou *"MATLAB Visualization"* a página é redirecionada para a central de modelos disponíveis, como vistos na figura abaixo.

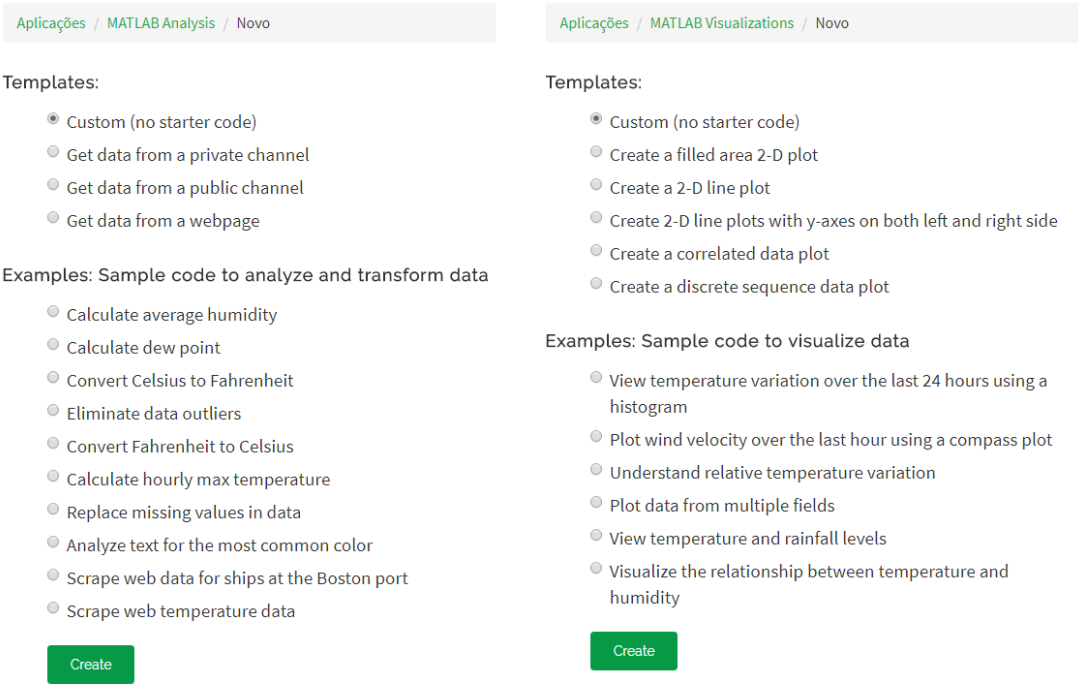


Figura 11: na figura a) pode-se visualizar as opções de modelo de obtenção de dados no MATLAB através da plataforma para efetuação de análise, enquanto na figura b) visualiza-se modelos de visualizações de dados.

Alguns exemplos são providenciados, como uma espécie de tutorial, portanto no modo análise pode-se obter exemplos de como calcular o valor médio, mínimo e máximo em um certo intervalo de tempo, como visto na figura 11 a) acima.

Além dos modelos de exemplo do modo análise, a ferramenta conta com exemplos de modelo do modo visualização também. Esse possui exemplos de disposição de gráficos mais elaborados, como por exemplo, criações de histogramas, gráficos com múltiplas entradas de dados e gráficos de correlações de dados, como vistos na figura 11 b) acima.

4 Metodologia

4.1 Materiais

Para a realização desse projeto serão necessários os seguintes materiais abaixo.

- 2 Sensores ultrassônicos HC-SR04.
- 2 Arduinos Uno R3.
- 1 Módulo ESP8266-01.
- 2 Módulos XBee S2C ZigBee (com antena).
- 2 Adaptadores de pinos para XBee.
- 2 Buzzers.
- 2 LED’s RGB.
- 8 Resistores 1k.
- 2 Reguladores de tensão 3,3 volts.
- 2 Fontes de alimentação de 5 volts.
- 2 Reservatórios para líquido.
- 2 Suportes para os sensores.
- Fios.
- 2 Placas ilha para prototipagem.
- Solda e ferro de solda.

4.2 Métodos

A construção do projeto foi dividida em duas partes, sendo elas a lógica e a física. A parte lógica consistirá da programação do Arduino, na linguagem de programação C, através da *IDE* própria do mesmo.

Através da programação do Arduino, serão recebidos os dados dos sensores ultrassônicos (para medição de nível volumétrico) que serão tratados a fim de obter a distância do sensor até o líquido e gerar uma taxa de elevação para que possa ser calculado o nível de criticidade do líquido.

Um dos módulos é chamado de *“coordinator”* e o outro de *“router”*. A nomenclatura se deve por conta do modo de comunicação na rede zigbee (vide figura 6). A diferença entre os módulos é apenas logicamente, e o fato de que o módulo

coordinator, tem saída para a internet, sendo necessário o envio dos dados do módulo *router* até o *coordinator*, por meio do XBee. Para melhor entender a topologia descrita acima, pode-se visualizar a figura abaixo.

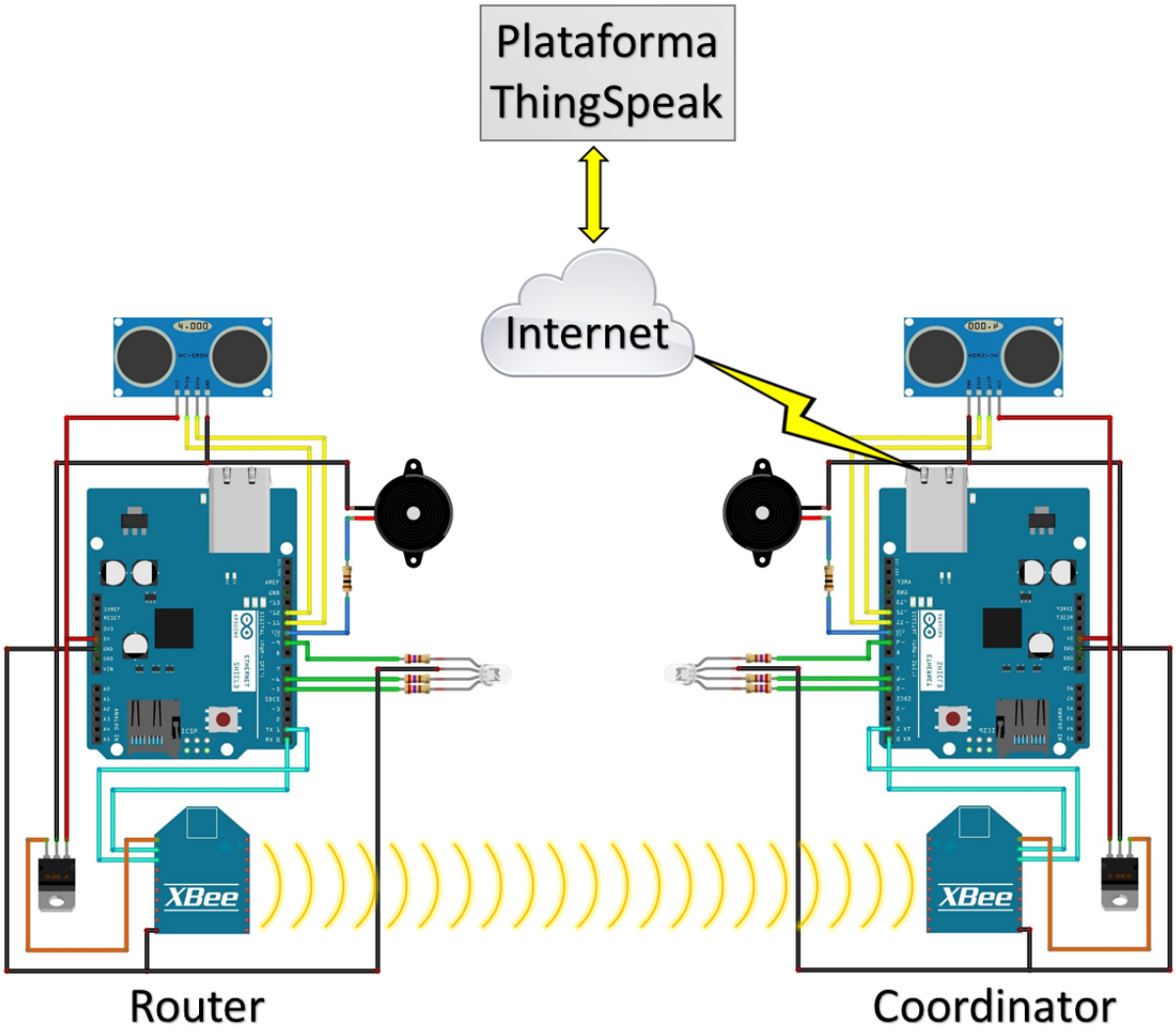


Figura 12: Representação da topologia total do projeto, simbolizando o módulo *router* se comunicando com o módulo *coordinator*, através do XBee, via rádio frequência, que através de sua comunicação com a internet no módulo *coordinator*, consegue realizar o envio de dados à plataforma ThingSpeak.

O diagrama de conexões dos arduinos pode ser visto abaixo na figura abaixo, exemplificando exatamente os pinos utilizados pelos diferentes componentes para se comunicarem com o arduino.

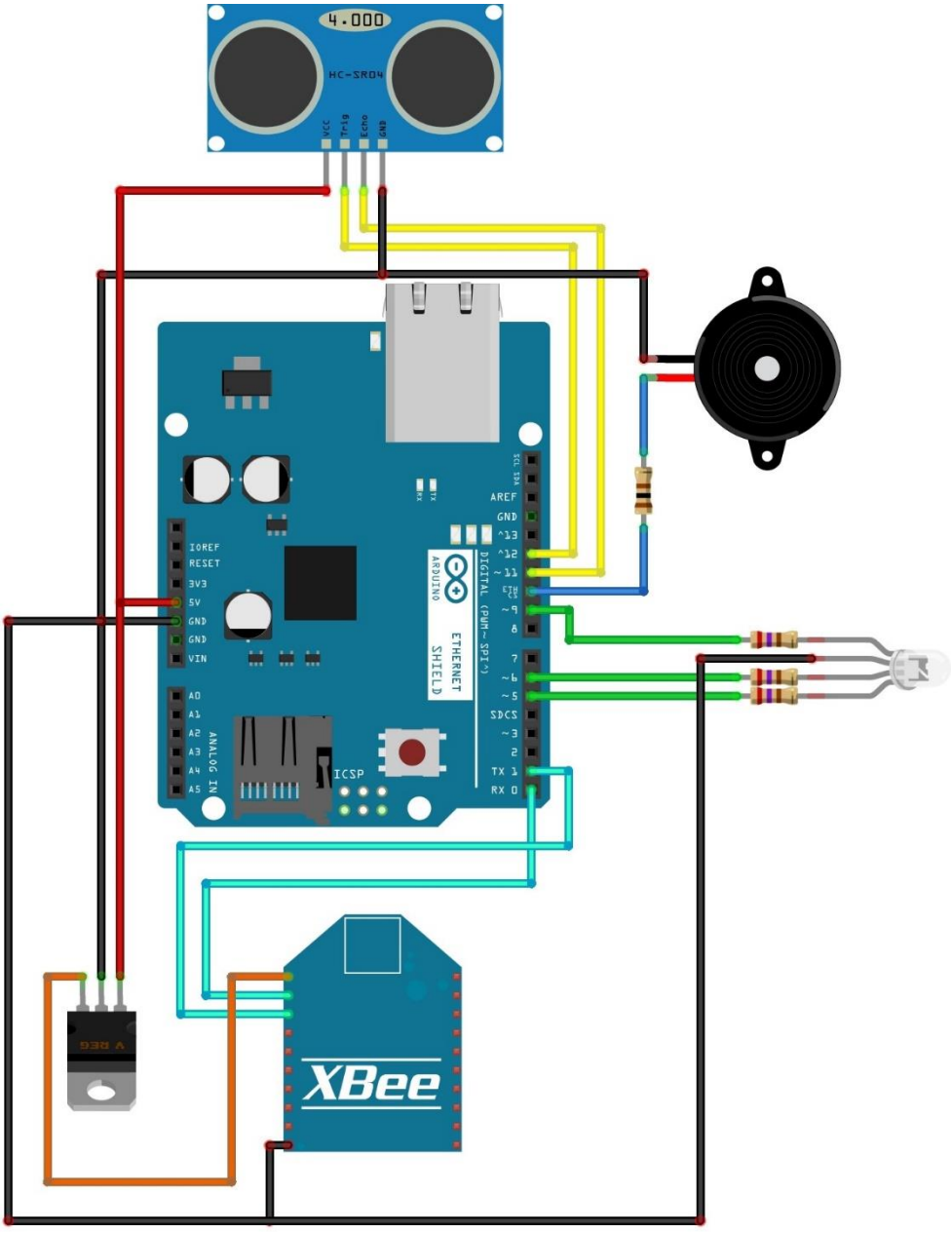


Figura 13: Diagrama de conexões dos módulos *coordinator* e *router*. Não faz parte do diagrama o cabo ethernet (diferencial físico entre os dois módulos).

a) b)

Figura 14: Módulos soldados para utilização final. Na figura a) pode-se vera parte superior do shield, enquanto que na figura b) pode-se ver a parte inferior do *shield*. Os módulos *coordinator* e *router* não diferem entre si no quesito de prototipagem.

O sensor foi colocado em um suporte de madeira criado para esse projeto. Desse modo o sensor é centralizado com o reservatório ficando a uma distância total de aproximadamente 50 centímetros do seu fundo. A representação mantida as proporções do sensor e reservatório pode ser vista na figura abaixo, onde os vértices do triângulo, no topo da imagem, simbolizam o início da cabeça do sensor.

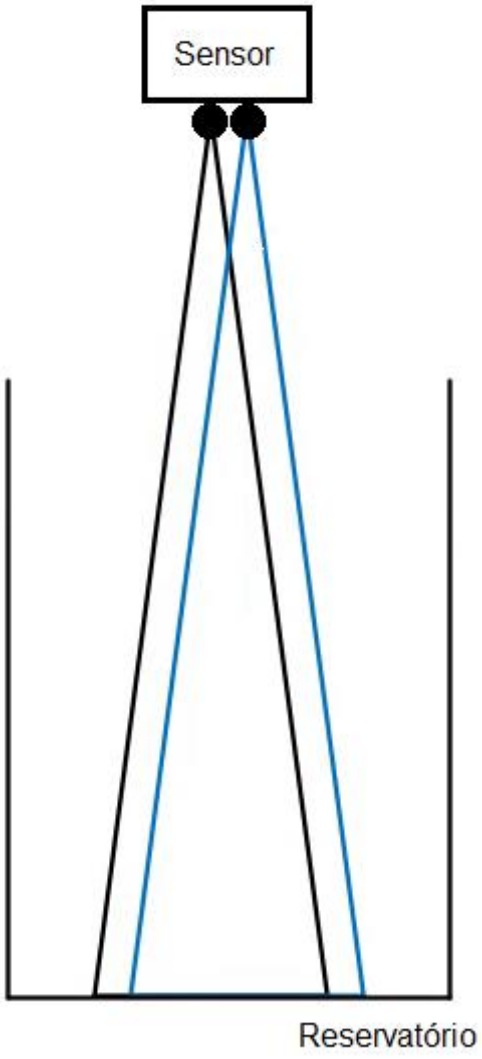


Figura 15: Representação em perfil do ângulo de operação do sensor e a sua distância do reservatório (mantidas as proporções). Os vértices dos triângulos indicam o início das duas cabeças do sensor, que formam um cone de operação, aumentando sua base à medida que a distância aumenta. Foi inserida uma diferenciação de cor para simbolizar as cabeças de *trigger* e *echo* do sensor.

Os cálculos de operação do sensor foram realizados para que não houvesse problemas de detectar um obstáculo e esse ser a lateral do reservatório.

Para o caso do módulo *coordinator*, a única diferença física está na presença do *shield* ethernet, fato esse que não altera nenhum esquema de ligação, inclusive a pinagem do arduino.

Nas figuras abaixo é possível ver o aparato criado para realizar o experimento.

a) b)

Figura 16: Aparato criado para realização do projeto. O sensor ficava apoiado no topo, virado para o fundo do reservatório.

O sensor ficava a aproximadamente 55 centímetros do reservatório e o balde possui pouco mais de 40 centímetros de altura.

Com o shield ethernet é possível enviar as informações para uma plataforma que foi desenvolvida visando a coleta e concentração de dados em tempo real, chamada *ThingSpeak*, para que sejam exibidos o nível volumétrico e a criticidade do nível medido. Essa comunicação entre o Arduino e a plataforma *ThingSpeak* seria realizada através de uma conexão Wi-Fi (módulo ESP8266), porém, durante o projeto optou-se pelo *shield* ethernet, por conta da maior estabilidade durante os testes. O projeto foi arquitetado visando uma topologia concentradora, sendo que apenas um Arduino teria acesso à *internet* e todos os dados são enviados para esse arduino central (*coordinator*), analogamente a um roteador em uma rede convencional. A conexão ponto-a-ponto inter-arduino foi realizada através de módulos XBee.

A construção da parte física envolve a prototipagem de 2 circuitos que ligarão o sensor, o XBee, shield ethernet, LED e *buzzer* ao Arduino, essas conexões foram soldadas para evitar mau contato. Inicialmente os testes foram realizados em uma protoboard, para facilitar as conexões.

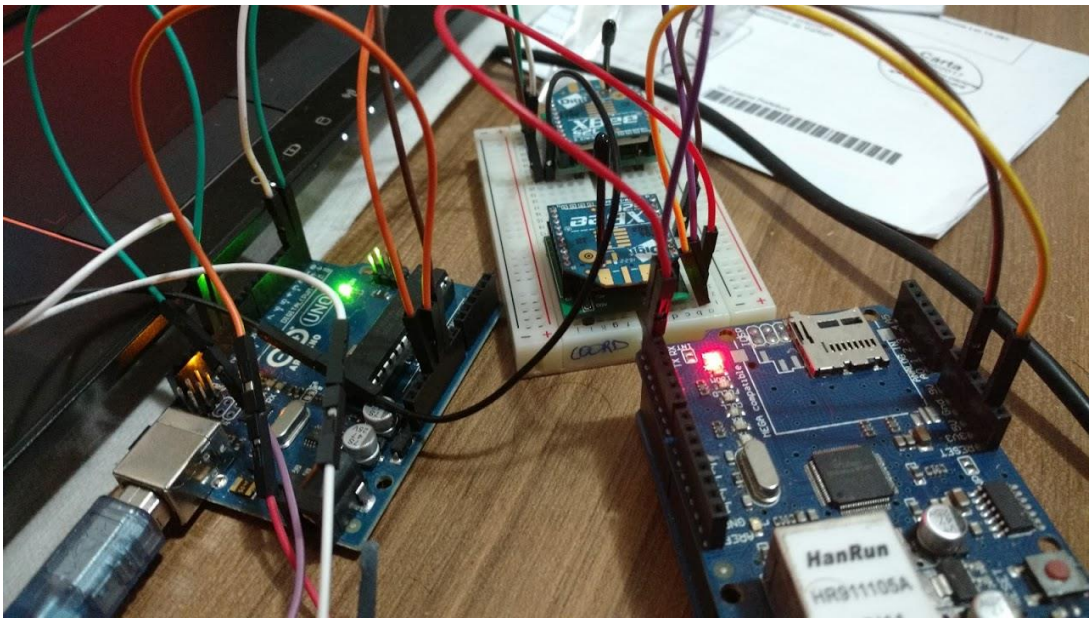


Figura 17: Testes iniciais do projeto utilizando a protoboard.

4.2.1 Configuração do XBee

A configuração dos dois módulos XBee foi realizadia com a conexão do módulo ao computador, utilizando um adaptador USB.

Uma vez que o módulo estava conectado e teve seu *driver* instalado, foi possível acessar o modo de configuração do módulo através da porta COM do computador. Com auxílio do *software* XCTU, os parâmetros necessários para efetuar a rede *mesh* do *Zigbee* puderam ser configurados via interface gráfica.

Um dos atributos configurados, foi o "*ID PAN ID*", que utiliza um número hexadecimal para configurar um número identificador de rede, ou seja, todos os módulos que devem comunicar-se entre si, precisam necessariamente ter o mesmo *ID* de rede. Para o projeto em questão, foi utilizado o *ID* "FF".

Outros atributos configurados foram o *DH* e *DL*, que significam *Destination Address High* e *Destination Address Low*, respectivamente. Esses parâmetros podem ser comparados com endereços MAC de redes convencionais, que são identificadores físicos e únicos de uma interface de comunicação, sendo uma parte responsável por identificar o fabricante, e a outra por identificar o equipamento.

Um atributo que não é vital para a comunicação com sucesso, porém facilita a gerência dos módulos, é a configuração do campo *NI Node Identifier*, que serve como nome do módulo em questão, como se fosse o *hostname* do equipamento.

Uma vez que esses parâmetros básicos foram configurados é necessário eleger um dos módulos como *coordinator*, para realizar tal eleição há um campo no *software* que se chama “*Coordinator Eligible*”, e pode-se alterar o valor entre 0 e 1, ou seja, foi selecionado um módulo para ter o valor 1 sendo ele o *coordinator* da rede *Zigbee*) e o outro módulo para ter o valor 0 (sendo ele um *router* da rede *Zigbee*).

Nas figuras 12 e 13 abaixo, é possível verificar a configuração do módulo *coordinator* e nas figuras 14 e 15 a configuração do módulo *router* através do *software XCTU*.

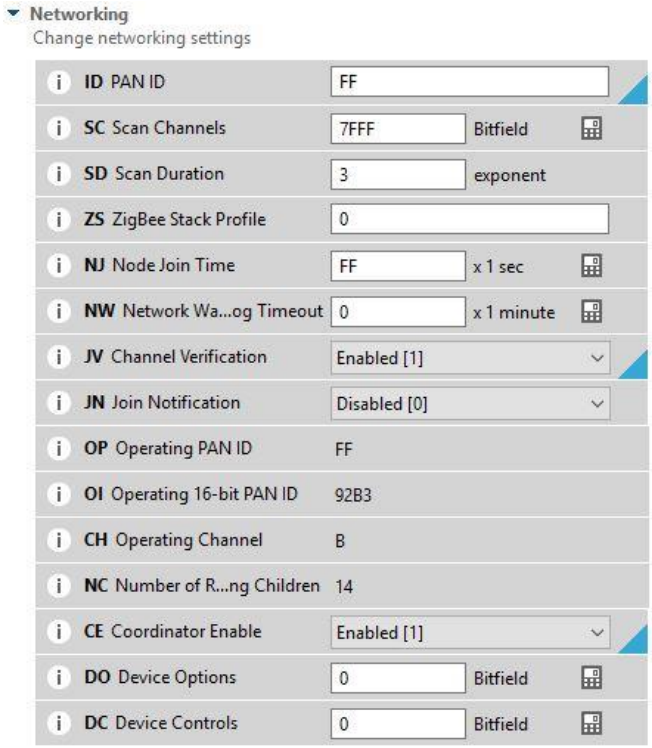


Figura 18: Seção de configuração de rede para o módulo coordinator do XBee.

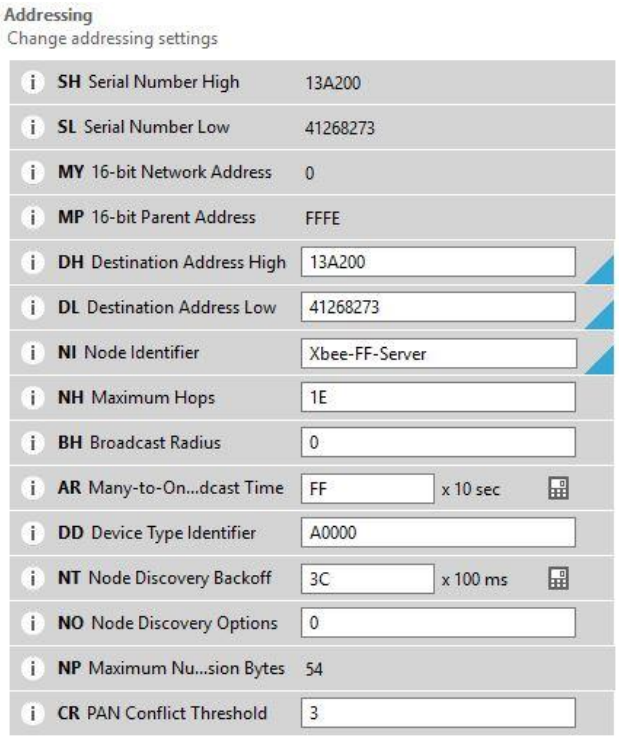


Figura 19: Seção de configuração de endereçamento para o módulo coordinator do XBee.

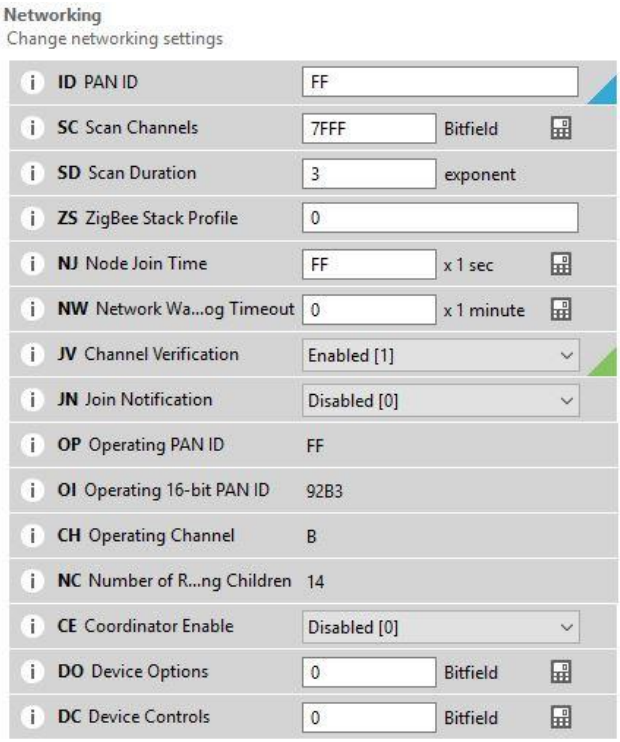


Figura 20: Seção de configuração de rede para o módulo router do XBee.

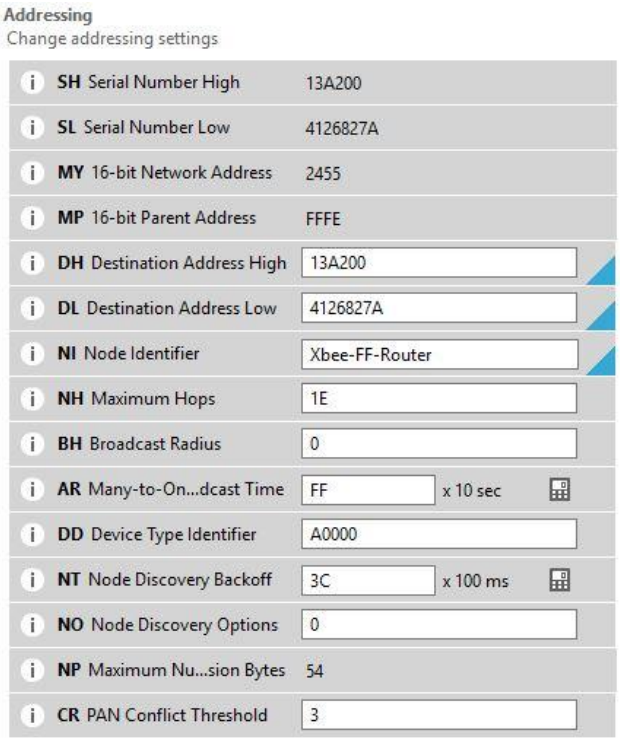


Figura 21: Seção de configuração de endereçamento para o módulo router do XBee.

4.2.2 Configuração do Arduino

4.2.2.1 Sensor

Os sensores foram conectados aos arduino de acordo com a figura 13. Para a sua configuração lógica foi necessário importar a biblioteca *NewPing*, definir os pinos do arduino para o *trigger* e o *echo* do sensor, executar o método *ping()* para obter o valor em microssegundos do *echo*, e executar o método *convert_cm()* para converter o tempo de deslocamento em uma distância em centímetros.

No anexo A é possível ver o código completo utilizado no projeto.

4.2.2.2 XBee

Foi realizada a ligação do XBee com o Arduino, seguindo o diagrama da figura 10. A comunicação ocorre através da escrita serial do código. Tudo que é direcionado para a saída serial, é transmitido do módulo *router* para o módulo *coordinator*.

Antes de comunicar um Arduino com o outro diretamente,foram realizadostestes comunicando um Arduino com o módulo *coordinator* conectado diretamente com o computador. Esse teste foi realizado através do console com o *software* XCTU fornece, sendo possível testar o envio de caracteres de um lado da comunicação para o outro.

4.2.2.3 Thingspeak

O Arduino que possui o módulo *coordinator* conectado, possui a saída para a internet. Com isso é possível realizar a configuração do *dashboard Thingspeak* (plataforma dedicada à exibição dos dados).

Para realizar essa configuração é necessário realizar um cadastro na plataforma e criar um canal de visualização (nomenclatura utilizada para a criação de uma página privada na plataforma). Cada canal possui sua própria chave de leitura e escrita, chamada de *apiKey*.

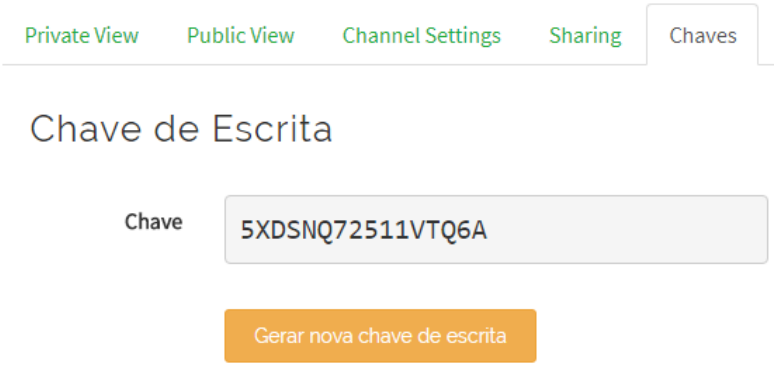


Figura 22: A chave de escrita pode ser encontrada dentro da seção de "chaves" do seu canal.

Figura 23: Embaixo da chave de escrita, está a chave de leitura, que pode ou não ser permitida, dependendo da política do seu canal.

Essa chave tem como objetivo autenticar a solicitação de escrita na plataforma (envio de dados via arduino), ou leitura de dados existentes (obtenção remota de dados para tratativa, como por exemplo, importação para o MATLAB). A chave de escrita é vital para o correto funcionamento e necessita estar inserida na configuração do Arduino para que o mesmo consiga autenticar-se com a plataforma.

O principal método envolvendo a plataforma *ThingSpeak* é o *updateThingSpeak()*, que pode ser visto em sua íntegra no anexo A.

Na figura abaixo é possível verificar o fluxograma do módulo *coordinator*. Após as definições de variáveis, e instanciamentos de bibliotecas, é executado o método *startEthernet()* que inicia a obtenção de endereço IP na rede através do protocolo DHCP. Caso o método seja executado com sucesso, e seja obtido conexão com a internet, é checado se o contador excedeu o intervalo de atualização, caso não tenha sido ainda excedido o intervalo, o sensor calcula a distância até líquido e em seguida é executado o cálculo de criticidade. Uma vez que é excedido o tempo de atualização, envia-se os dados à plataforma.

Depois de enviar os dados à plataforma, é checado se continua conectado à internet. Se por um motivo a conexão for perdida, é tentado a reconexão por três vezes antes de aguardar um período.

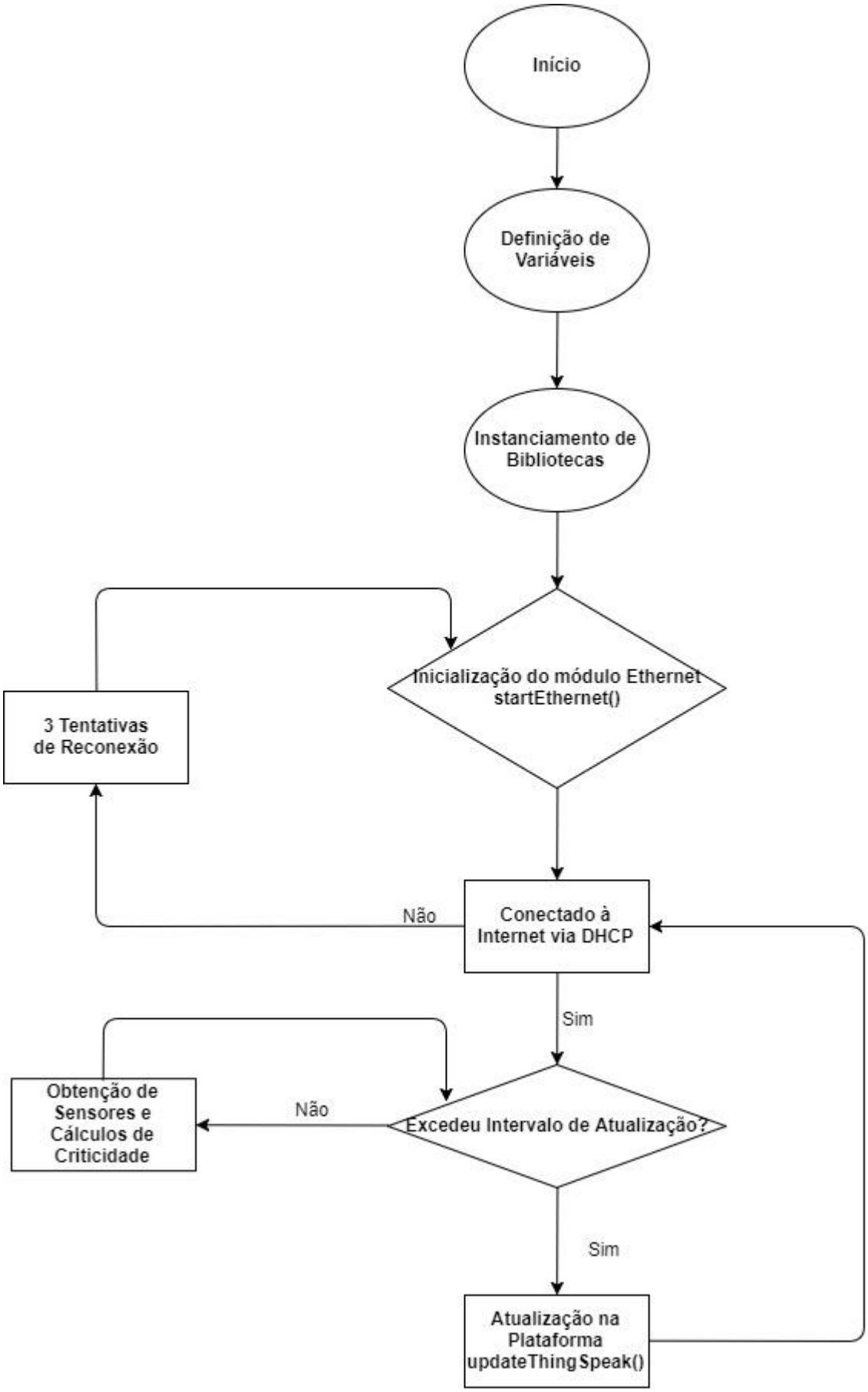


Figura 24: Fluxograma com a operação do código.

5 Resultados e Discussão

5.1 Sensor

Depois de obtido os valores de distância do sensor ao líquido, foi possível realizar tratativas e manipulações com os mesmos.

Calcula-se a razão de subida do nível do líquido através da equação 2 abaixo.

$$Razão = \frac{Nível\ antigo}{Nível\ atual} \qquad (2)$$

A distância calculada não é o nível do líquido em si, e sim a distância do sensor ao líquido, por isso utiliza-se a razão do “nível antigo” pelo “nível atual”. Por exemplo, inicialmente o reservatório encontra-se vazio, portanto a variável “nível atual” assume o valor 55, na próxima medição, o a variável “nível antigo” assume o valor do nível atual, e a variável nível atual assume um novo valor, por exemplo 50 (indicando uma inserção de 5 centímetros de coluna de água no reservatório). Desse modo, utilizando a equação 2, obtém-se:

$$Razão = \frac{55}{50} \Rightarrow Razão = 1{,}1$$

Através desse cálculo acima, entende-se que o nível volumétrico como um todo aumentou 10% na medição atual em relação a medição antiga.

Com esse cálculo foi possível determinar a taxa de aumento ou diminuição do nível volumétrico. Com a taxa calculada, e levando em consideração a distância do sensor ao líquido, ou seja, quão próximo de um transbordo está, foi elaborada uma matriz de criticidade, que é possível ver na tabela 2 abaixo, seguida de sua explicação.

Tabela 2: Tabela de criticidade do nível volumétrico levando em consideração a razão entre as últimas duas medições de nível, e a distância atual do sensor ao nível, sendo 5 o mais crítico e 1 o menos crítico.

		Razão “R” entre as últimas duas medições do sensor				
		R < 1	1 ≤ R < 1,1	1,1 ≤ R < 1,2	1,2 ≤ R < 1,3	R ≥ 1,3
Distância “X” do sensor	X ≥ 40	1	1	1	2	3
	25 ≤ X < 40	1	2	2	3	4

	15 ≤ X < 25	2	2	3	4	4
	X < 15	3	3	4	5	5

Essa matriz foi confeccionada para atribuir pesos diferentes a situações diferentes. Como o reservatório possui uma altura de aproximadamente trinta e cinco centímetros, estipulou-se uma primeira faixa de distância a partir de quarenta centímetros do sensor. Escolheu-se essa distância para os casos em que o reservatório se encontra vazio, sem líquido no momento. O segundo intervalo proposto é entre vinte e cinco e quarenta centímetros do sensor, para os casos em que o líquido começou a preencher o reservatório, porém não ultrapassou a metade do mesmo. A terceira faixa de operação proposta foi entre dez e vinte cinco centímetros do sensor. Nesse caso, o reservatório continua enchendo e ultrapassou a metade do volume do reservatório. O último intervalo de distância proposto foi à dez centímetros do sensor, valor esse que representa o reservatório totalmente preenchido pelo líquido.

De acordo com a equação 2, descrita acima, foi calculada a razão do nível volumétrico, ou seja, é calculada a taxa de regressão (quando menor que um) ou aumento (quando maior que um). Se a taxa for igual a um, indica que o nível anterior é exatamente igual ao nível atual medido. Assim foram estipulados os valores de R da tabela de modo a cada coluna simbolizar respectivamente um regresso no nível volumétrico, um aumento de até 10%, um aumento de até 20%, um aumento de até 30% e aumentos acima de 30% no nível volumétrico.

Com as informações de intervalo de distância do líquido ao sensor, e a taxa de variação do líquido, foi estipulada a tabela 2 acima, com valores de criticidade entre um intervalo entre um e cinco, sendo um o caso mais crítico e cinco o caso menos crítico. A determinação de qual valor de criticidade cada célula da tabela obteria foi efetuada com base nos mínimos e máximos, ou seja, uma ponta da tabela é menos crítica, enquanto a criticidade vai aumentando diagonalmente até a outra ponta, onde encontra-se o máximo.

Uma distância de 10 centímetros do sensor é menos crítica em uma situação onde começa a haver uma regressão, do que em uma situação que ocorre um

aumento de 30%, em relação a última medição. No código, a função que executa esse cálculo pode ser encontrada como *critical()*.

O esquema de cores representados na tabela 2 simbolizam as cores que o LED RGB exibe em cada criticidade medida.

Os valores foram adaptados para o teste realizado em pequena escala, portanto seria necessário adaptar os valores para um caso de teste real, monitorando um rio, por exemplo.

Uma vez que a criticidade foi calculada, ela é utilizada no projeto para definir a cor do LED indicador a acender, e a execução ou não de um *buzzer* para alertar a situação.

5.2 XBee

A comunicação através do XBee ocorreu corretamente, sem perdas visíveis na transmissão. Ao enviar o dado de um módulo ao outro através da comunicação serial, é enviado um byte por vez, por consequência, do lado receptor chega-se o código decimal da representação da tabela ASCII, sendo necessário executar a sua conversão para um caractere.

Após a obtenção do número como um caractere, é executada uma nova mudança de tipo de variável, transcrevendo-a para um inteiro, para que possa ser realizada as manipulações necessárias. A partir dessa mudança foi possível manipular os dados para executar os cálculos da taxa de subida de nível e criticidade.

5.3 Comunicação com Internet

Para o funcionamento, foi necessário configurar um MAC *Address* único para o arduino, e a obtenção de endereço IP se dá através do protocolo DHCP.

Percebeu-se a necessidade durante o projeto de utilizar uma fonte externa à da alimentação via USB (mantendo um terra comum com relação à saída de 5 volts do arduino), visto que os módulos utilizados podem acabar necessitando de maior energia que o USB conectado ao computador pode prover. Para isso, pretende-se usar um adaptador de fonte externa de 5 volts e 2 amperes, fornecendo potência suficiente.

5.4 ThingSpeak

Durante um período de uma hora de testes, com intervalos de 16 em 16 segundos (tempo mínimo na plataforma grátis), não foi perdida nenhuma atualização na plataforma, conforme exemplo de figura abaixo.

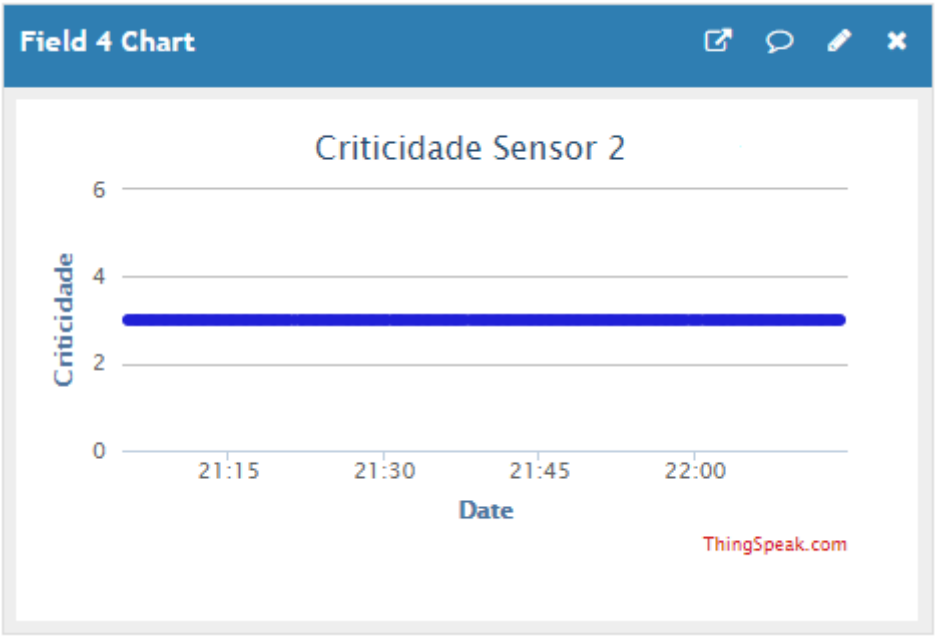


Figura 25: Atualização de campo na plataforma durante um período de uma hora com atualizações regulares de 16 segundos.

A atualização de dados na plataforma é executada através do comando abaixo.

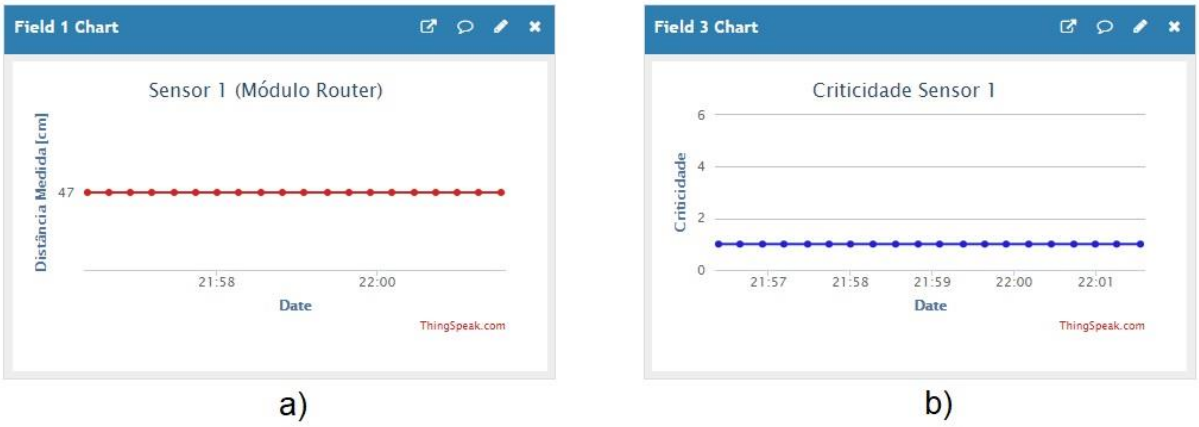
```
1. updateThingSpeak("field1="+phrase+"&field2="+String(cm)+"&field3="+criticidadeSensor
   1+"&field4="+criticidadeSensor2);
```

Para a correta atualização de informações na plataforma, são concatenados os valores a serem enviados à plataforma precedido do campo a ser enviado (*field1*, *field2*, *field3* e *field4*).

O método invocado abre um soquete com o servidor que autentica a chave de escrita (*api*), e autoriza o recebimento dos dados.

Uma vez que os dados são enviados, eles são automaticamente plotados num gráfico, em função do tempo.

Como um primeiro teste, foi deixado os reservatórios vazios durante um intervalo, para controle de modo a testar a estabilidade. Como não houve mudanças no nível, o resultado obtido está de acordo com o esperado, uma constante.



a) b)

Figura 26: Testes de estabilidade de medição para o sensor 1 para o primeiro teste.

a) b)

Figura 27: Testes de estabilidade de medição para o sensor 2 para o primeiro teste.

Enquanto a criticidade permanece constante como esperado em ambos os casos representados nas figuras acima, no gráfico a) da figura 24 há uma leve variação, oscilando um centímetro para mais e para menos, ao redor dos cinquenta centímetros esperados para o teste. Acredita-se que essa variação não é significativa e não tenha impacto negativo para os testes, visto que essa variação representa apenas 2% do valor total. Essa variação pode ser causada por vários fatores, desde imperfeição do sensor, posicionamento ruim, ou algum tipo de imperfeição nas conexões.

Abaixo é possível visualizar um segundo teste onde foi enchendo-se o reservatório gradativamente e depois removido um pouco do líquido no final.

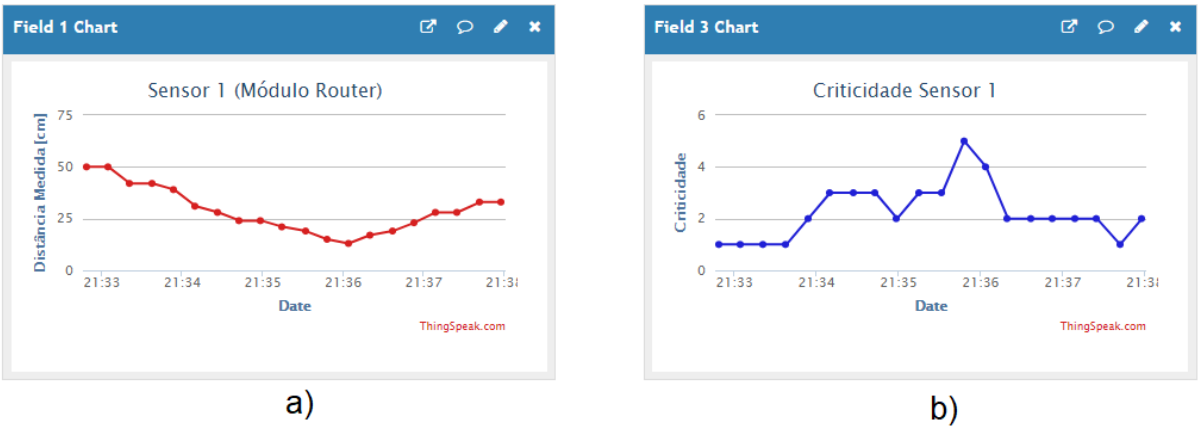


Figura 28: Gráfico da distância do sensor 1 ao líquido em função do tempo na figura a) e da sua criticidade calculada na figura b) para o segundo teste.

A medida que o reservatório começa a se encher pode-se notar a criticidade aumentando junto. Na figura acima, percebe-se por volta de 21:35 que houve um momento em que o nível se manteve, o que, segundo a tabela de criticidade, é um bom sinal, fazendo com que o valor da criticidade voltasse para 2 invés de 3 na figura b) acima.

Quando o ápice do teste é atingido por volta de 21:36 (mínima distância entre o sensor e o líquido do reservatório), atinge-se a criticidade máxima, cujo valor é cinco, conforme visto na figura b) acima. Assim que se inicia o esvaziamento do reservatório, percebe-se uma diminuição esperada na criticidade.

A estabilidade no nível do líquido indica uma situação menos crítica que um preenchimento abrupto, portanto, com o preenchimento aproximadamente constante, é gerado um efeito de criticidade aumentando devagar. Já o esvaziamento do reservatório causa um efeito rápido de criticidade baixa, indicando uma situação mais confortável.

No terceiro teste abaixo, foi realizada o preenchimento abrupto do reservatório, e depois o esvaziamento abrupto, e repetida a operação.

Figura 29: Gráfico da distância do sensor 1 ao líquido em função do tempo na figura a) e da sua criticidade calculada na figura b) para o terceiro teste.

Percebe-se que enquanto o nível estava estável (no começo), a criticidade estava baixa (medindo um). Entre o terceiro e o quarto ponto da figura a) acima houve um preenchimento abrupto, fazendo com que o sistema ficasse crítico rapidamente (como esperado), indo para o valor 5 na figura b) acima. Como o nível se manteve depois de preenchido o reservatório a criticidade diminuiu de patamar, assumindo o valor 3. Durante esvaziamento abrupto, a razão entre as duas últimas medidas se torna menor do que 1, indicando o esvaziamento, e por consequência, assumindo a primeira coluna da tabela 2, de menor risco, fazendo com que a criticidade retornasse para o valor um. O código trata o esvaziamento como algo bom, fazendo com que a criticidade diminua mais do que em outros casos.

Na metade do teste, quando é retornada à mesma posição inicial executa-se o mesmo teste, com intuito de obter o mesmo resultado. Porém a criticidade na figura b) nunca chega ao valor cinco. Isso se deve ao fato de uma imperfeição na medição quando ocorre o preenchimento abrupto, no gráfico a) pode-se ver que o primeiro ponto após a segunda queda (por volta de 22:18) não é igual aos pontos seguintes, com isso, esse primeiro ponto marcava a medida 15 e com isso não entra no último intervalo de medições proposta a tabela 2. Na medição seguinte, ele assume o valor 14, porém como a mudança de 15 para 14 é singela, a criticidade (de acordo com a tabela 2) nunca assume o valor 5.

No quarto teste realizado, foi simulado um ambiente que ocorre um preenchimento rápido do reservatório, feito isso o nível se mantém por algumas medições, e então ocorre o esvaziamento do mesmo.

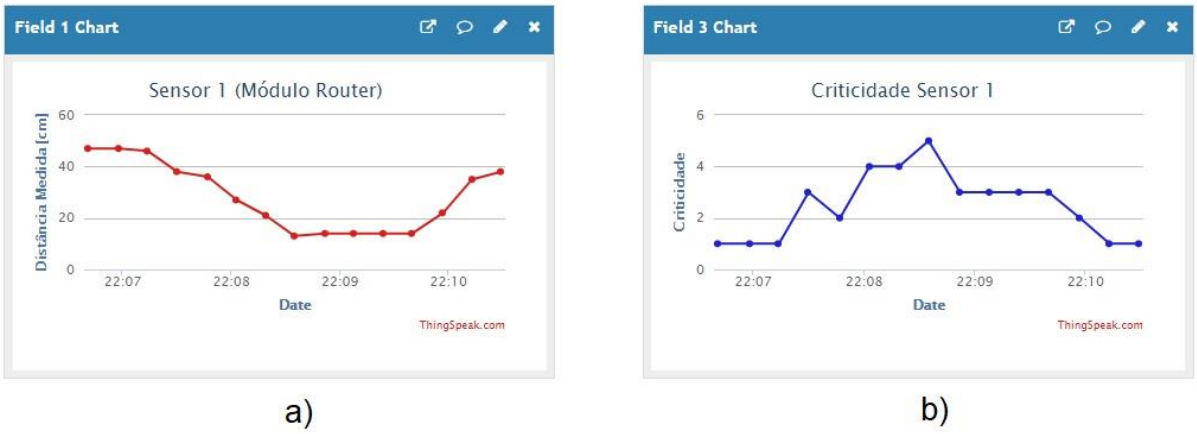


Figura 30: Gráfico da distância do sensor 1 ao líquido em função do tempo na figura a) e da sua criticidade calculada na figura b) para o quarto teste.

Entre o quarto e quinto ponto da figura a) acima, nota-se uma diminuição na taxa de preenchimento, isso faz com que a criticidade retraia de três para dois. Nas próximas medições, resume-se a taxa inicial, fazendo com que a criticidade volte a subir, até atingir o ápice entre 22:08 e 22:09, momento esse que é o primeiro ponto de menor medição da figura a). À medida que a o nível permanece constante, nota-se a criticidade retornando ao valor três, e mantendo-se assim até que se inicia o esvaziamento do reservatório.

Para o quinto, e último teste realizado, foi efetuado novamente o preenchimento e esvaziamento do reservatório, de modo gradativos, porém, nesse teste, foram utilizados mais pontos a fim de confeccionar um gráfico com mais dados para o segundo sensor.

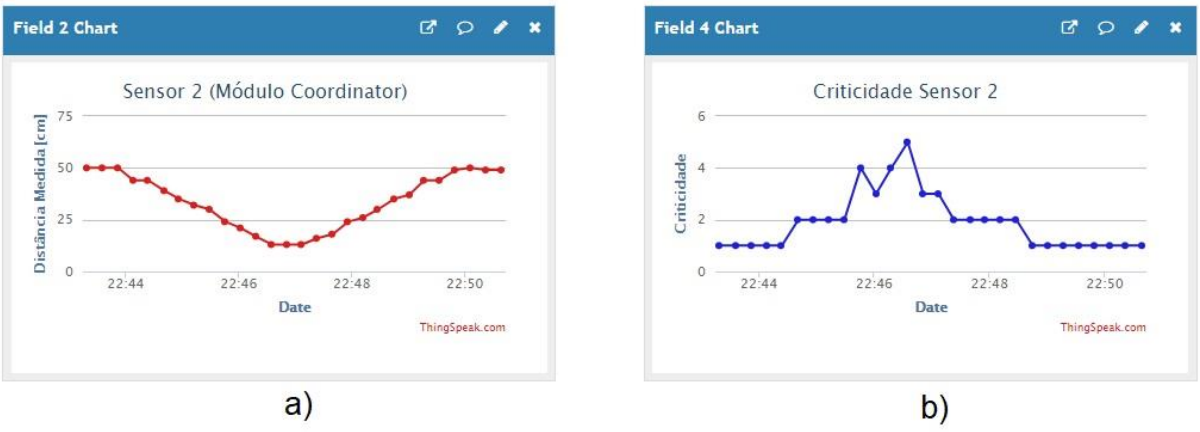


Figura 31: Gráfico da distância do sensor 2 ao líquido em função do tempo na figura a) e da sua criticidade calculada na figura b) para o quarto teste.

A medida que o reservatório começa a ser preenchido, vai aproximando-se do limiar para atingir valor dois para a criticidade, como as variações são pequenas, esse valor é atingido entre 22:44 e 22:46, conforme visto nas figuras a) e b) acima. Conforme a distância ao sensor vai diminuindo, a criticidade vai aumentando, porém, há um ponto da figura b) em que ocorre uma diminuição da criticidade quatro para três, depois retornando para quatro e cinco em seguida. Isso se deve a variação da taxa da equação 2, pois o valor entre uma medição não obteve um intervalo muito significativo, de modo caracterizar a criticidade com um valor menor, isso pode ser visto no ponto próximo a 22:46 da figura a). A partir do ponto médio da figura a), onde o nível se estabiliza em seu ápice e em seguida começa o esvaziamento, observa-se na figura b) uma diminuição da criticidade conforme esperada.

6 Conclusão

Acredita-se que a escolha do sensor foi justificada pelos resultados. Ele gerou dados confiáveis e sem muita oscilação. Havia um certo receio com sua utilização pois a sua escolha foi dada principalmente pelo seu baixo custo. A outra opção, era um sensor diferencial de pressão, que calculava a pressão do ar, e a pressão da coluna do líquido, através de uma mangueira inserida no reservatório, porém o seu preço era mais elevado (por volta de 70 reais, e o sensor ultrassônico 5 reais), e pensando no objetivo final de aplicação em um rio ou córrego, esse sensor diferencial era menos prático a utilização, sendo que pela alta presença de detritos, a medição poderia ser menos confiável.

O Zigbee foi escolhido por conta dos protocolos voltado à IoT, porém, percebe-se no projeto que a utilização de módulos de comunicação RF de 433 MHz serviriam para esse propósito, e sairia mais barato o projeto como um todo. Um módulo Zigbee pode ser encontrado por volta de 120 reais, e os módulos de transmissão e recepção de 433 MHz podem ser adquiridos por volta de 20 reais. Realizando testes futuros, possa consolidar a opção de uso do Zigbee, visto que segundo seu datasheet, o alcance máximo é em torno de 1200 metros.

Com base nos resultados dispostos no capítulo cinco acredita-se que, para os testes realizados e ambiente proposto, os resultados são significativos e corroboram a possibilidade de seguir adiante com o projeto. Em alguns casos, pode-se realizar uma melhora do código de modo aperfeiçoar o cálculo de criticidade. Porém os resultados se mostraram conforme esperados, seguindo os valores propostos na tabela 2. A medida que ocorre um preenchimento do reservatório, e atinge o limiar, a mudança de criticidade ocorre corretamente. Quando é atingido o equilíbrio volumétrico a criticidade varia rapidamente, melhorando, porém, em alguns testes realizados pode apenas significar que o preenchimento do reservatório não foi tão rápido quanto a medição do sensor.

O ajuste fino do código de cálculo da criticidade seria necessário, visto que caso ocorra um cenário onde o nível fica oscilando levemente entre um valor maior que o outro, o nível de criticidade ficará oscilando também.

Conclui-se, portanto, que o projeto é viável para execução de testes em larga escala (apesar de ter que ajustar os valores para um ambiente maior), pois mesmo para um ambiente pequeno e controlado os dados obtidos eram críveis e com uma

precisão boa. No projeto foi utilizado um tempo de atualização pequeno por conta dos testes, mas com pequenos ajustes acredita-se ser possível um teste maior, em um ambiente real.

No caso de levar para testes em um ambiente real, acredita ser necessário aumentar o intervalo de atualização na plataforma, pois o mesmo foi posto de 16 em 16 segundos para facilitar os testes, não sendo necessário uma atualização tão frequente para a medição de um rio, propriamente dito.

A plataforma utilizada tem mais funcionalidades a oferecer do que foi utilizada no momento (como a integração com o MATLAB) porém, para trabalhos futuros, seria interessante uma plataforma que armazenasse os dados antigos, para que possa iniciar uma segunda etapa de análise, para tentativa de predição de evento, possivelmente utilizando técnicas de RNA (redes neurais artificiais) e/ou outras técnicas de inteligência artificial para realizar predições de valores.

Foi encontrado problemas com a utilização do módulo ESP8266 por conta da baixa quantidade de informação disponível em fóruns, inclusive do fabricante. Outro problema encontrado foi referente à alimentação elétrica do módulo. O mesmo opera à uma tensão 3,3 V, porém o consumo de corrente desde 1 mA até 170 mA à medida que o módulo abandona o estágio de hibernação e assume o papel de transmissão de dados. No futuro, espera-se ser possível adicionar esse módulo ao projeto, de modo a não precisar de um cabo *ethernet* para a comunicação com a *internet*.

7 Referências

[1] Portal Brasil. **IBGE: Metade dos brasileiros teve acesso a internet em 2013.** 2014. Disponível em: <http://www.brasil.gov.br/infraestrutura/2014/09/ibge-metade-dos-brasileiros-teve-acesso-a-internet-em-2013>. Acesso em: 27 jun. 2015.

[2] NIC.BR. **IPv6:** Introdução. Disponível em: <http://ipv6.br/post/introducao/>. Acesso em: 07 ago. 2016.

[3] IBGE. **Perfil dos Municípios Brasileiros 2013.** 2014. Disponível em: <ftp://ftp.ibge.gov.br/Perfil_Municipios/2013/munic2013.pdf>. Acesso em: 20 ago. 2016.

[4] SANTO ANDRÉ. SEMASA. **Semasa: Saneamento Ambiental**. Disponível em: <http://www.semasa.sp.gov.br/index.php?page=monitoramento-cameras>. Acesso em: 07 ago. 2016.

[5] MIORANDI, Daniele et al. **Internet of things: Vision, applications and research challenges**. Ad Hoc Networks, v. 10, p. 1497-1516. set. 2012.

[6] TANENBAUM, Andrew Stuart; ANDREW TANNEMBAUM. **Computer networks.** 4. ed. Upper Saddle River, USA: Prentice Hall, c2003. xx, 891. ISBN 9780130661029

[7] EVANS, Dave. **The Internet of Things: How the Next Evolution of the Internet Is Changing Everything**. Cisco, 2011. Disponível em: <http://www.cisco.com/web/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf>. Acesso em: 27 jun. 2015.

[8] PEASE, Roland. **Internet of things: Trash talk signals mobile future.** 2014. Disponível em: <http://www.bbc.com/future/story/20130314-trash-talk-signals-mobile-future>. Acesso em: 27 jun. 2015.

[9] ATZORI, Luigi; IERA, Antonio; MORABITO, Giacomo. **The Internet of Things: A survey**. Computer Networks**,** v. 54, n. 15, p.2787-2805, out. 2010.

[10] WASHBURN, Doug et al. **Helping CIOs Understand "Smart City" Initiatives.** Forrester, 2010. Disponível em: <http://www-935.ibm.com/services/us/cio/pdf/forrester_help_cios_smart_city.pdf>. Acesso em: 27 jun. 2015.

[11] NAM, Taewoo; PARDO, Theresa. **Conceptualizing Smart City with Dimensions of Technology, People, and Institutions**.International Conference On Digital Government Research, n. 12, p.282-291, jun. 2011. Anual.

[12] ARDUINO. **Arduino:** ArduinoBoardUno. Disponível em: <https://www.arduino.cc/en/Main/ArduinoBoardUno>. Acesso em: 07 ago. 2016.

[13] DESCONHECIDO. **ESP8266:** Module Family. Disponível em: <http://www.esp8266.com/wiki/doku.php?id=esp8266-module-family>. Acesso em: 07 ago. 2016.

[14] KUMAR, Sandeep; GAGANDEEP. **A Study on Zigbee Architecture and Functional Model for WPAN Network**. International Journal Of Research In Management, Science & Technology, v. 3, n. 2, p.156-159, abr. 2015.

[15] EADY, Fred. **Hands-On ZigBee: Implementing 802.15.4 with Microcontrollers.** Elsevier, 2007.

[16] VALCK, Peter et al. **Exploiting programmable architectures for WiFi/ZigBee inter-technology cooperation**. Eurasip Journal On Wireless Communications And Networking, v. 2014, n. 1, p.212-224, 2014.

[17] EDUARDO TUDE (Org.). **Zigbee.** Disponível em: <http://www.teleco.com.br/tutoriais/tutorialzigbee/pagina_1.asp>. Acesso em: 07 ago. 2016.

[19] PATERSON, A. R; A.R. PATERSON. **A first course in fluid dynamics.** Cambridge, USA: Cambridge University Press, 1983. vii, 528. ISBN 9780521274241.

[20] ESPRESSIF SYSTEMS IOT TEAM. **ESP8266EX Datasheet.** Versão 4.3: Espressif Systems, 2015. 31 p. Disponível em: <http://download.arduino.org/products/UNOWIFI/0A-ESP8266-Datasheet-EN-v4.3.pdf>. Acesso em: 13 jul. 2017.

[21] ELEC FREAKS. **Ultrasonic Ranging Module HC - SR04.** Disponível em: <http://www.micropik.com/PDF/HCSR04.pdf>. Acesso em: 10 out. 2017.

8 Anexo A

8.1 Código do modulo Coordinator

```
1. // Bibliotecas
2. #include <SPI.h>
3. #include <Ethernet.h>
4. #include <NewPing.h>
5.
6. // Definicao de variaveis para o sensor ultrassonico
7. #define ECHO_PIN 11 // Pino do arduino onde conecta o pino "Echo" do sensor
8. #define TRIGGER_PIN 12 // Pino do arduino onde conecta o pino "Trigger" do sensor
9. #define MAX_DISTANCE 500 // Limitador de distancia maxima (em centimetros) para medir atraves do sensor
10.
11. // Configuracao de LAN. Ednereco IP sera atribuido via DHCP
12. byte mac[] = { 0xD4, 0x28, 0xB2, 0xFF, 0xA0, 0xA1 }; // Inicialização de MAC Address igual a D4:28:B2:FF:A0:A1
13.
14. // Configuracao ThingSpeak
15. char thingSpeakAddress[] = "api.thingspeak.com";      // Endereço da API do Thingspeak
16. String writeAPIKey = "5XDSNQ72511VTQ6A";              // Chave unica de escrita para o canal criado do Thingspeak
17. const int updateThingSpeakInterval = 20 * 1000;      // Intervalo de tempo (milisegundos) para atualização do Thingspeak (segundos * 1000 = intervalo = 20 segundos)
18.
19. // Inicializacao de variaveis
20. long lastConnectionTime = 0;
21. boolean lastConnected = false;
22. int failedCounter = 0;
23. char dec2Ascii;
24. int ascii2Int;
25. String phrase;
26. int sensor1Antigo = 1;
27. int sensor1Atual = 0;
28. int sensor2Antigo = 1;
29. int sensor2Atual = 0;
30. int criticidade1 = 5;
31. int criticidade2 = 5;
32. String criticidadeSensor1;
33. String criticidadeSensor2;
34.
35. // Inicializacao do modulo ethernet
36. EthernetClient client;
37.
38. // Instaciamento da biblioteca NewPing, passando os valores dos pinos de "Trigger", "Echo" e a maxima distancia a ser medida
39. NewPing sonar(TRIGGER_PIN, ECHO_PIN, MAX_DISTANCE);
40.
41. // Inicio do metodo setup
42. void setup()
43. {
44.   // Inicializacao do serial para debug
```

```
45.   Serial.begin(9600);
46.
47.   // Execucao do ethernet no arduino
48.   startEthernet();
49. } // Termino do metodo setup
50.
51. // Inicio do metodo loop
52. void loop()
53. {
54.   // Envio de um ping, atribui a resposta (tempo em micro segundos) na vari
    avel uS (tempo de ida e volta)
55.   unsigned int uS = sonar.ping();
56.   // Conversao do tempo de ida e volta para metros
57.   unsigned int cm = sonar.convert_cm(uS);
58.
59.     // Aquisicao de dados do modulo Router entquato estiver disponivel
60.   if (Serial.available() > 0)
61.   {
62.     // Atribuicao do dado obtido através do Xbee. A comunicação envia o val
    or em decimal referente ao número calculado no outro modulo
63.     int temp = Serial.read();
64.     // Conversao do valor em decimal recebido para o caracter ascii
65.     char dec2Ascii = temp;
66.     // Cast do caracter ascii para um string, para ser utilizado no metodo
    de update para o Thingspeak
67.     phrase = String(phrase+dec2Ascii);
68.     ascii2Int = phrase.toInt();
69.   }
70.
71.   // Print da resposta do update no Serial Monitor
72.   if (client.available())
73.   {
74.     char c = client.read();
75.     Serial.print(c);
76.   }
77.
78.   // Desconectado do Thingspeak
79.   if (!client.connected() && lastConnected)
80.   {
81.     Serial.println("...disconnected");
82.     Serial.println();
83.
84.     client.stop();
85.   }
86.
87.   // Atualizacao do ThingSpeak respeitando as condicoes de 1. Nao ter um cl
    iente conectado no momento, e o tempo da ultima conexao ser maior do que o
    intervalo de atualizacao
88.   if(!client.connected() && (millis() - lastConnectionTime > updateThingSpe
    akInterval))
89.   {
90.     sensor1Atual = ascii2Int;
91.     criticidade1 = critical((int)sensor1Atual, (int)sensor1Antigo);
92.     sensor1Antigo = sensor1Atual;
93.     criticidadeSensor1 = String(criticidadeSensor1+String(criticidade1));
94.
95.     sensor2Atual = (int) cm;
96.     criticidade2 = critical(sensor2Atual, sensor2Antigo);
97.     sensor2Antigo = sensor2Atual;
```

```
98.     criticidadeSensor2 = String(criticidadeSensor2+String(criticidade2));
99.
100.          // chamada do metodo para enviar os valores para a dashboard para
    os diferentes "field" disponiveis na plataforma
101.          updateThingSpeak("field1="+phrase+"&field2="+String(cm)+"&field3=
   "+criticidadeSensor1+"&field4="+criticidadeSensor2);
102.          // Reset da string prhase, para utilizar na proxima aquisicao de
   dados
103.          phrase = "";
104.          criticidadeSensor1 = "";
105.          criticidadeSensor2 = "";
106.        }
107.
108.        // Checagem se o modulo ethernet do arduino necessita ser reinicial
   izado
109.        if (failedCounter > 3)
110.        {
111.          // Execucao do ethernet no arduino
112.          startEthernet();
113.        }
114.
115.        // Atribuicao do valor booleano para a variavel lastConnected
116.        lastConnected = client.connected();
117.      } // Termino do metodo loop
118.
119.      // Inicio do metodo updateThingSpeak, passa-
   se uma string para o metodo
120.      void updateThingSpeak(String tsData)
121.      {
122.        // Se o cliente estiver conectado no endereco da api, na porta logi
   ca 80, executa o condicional abaixo
123.        if (client.connect(thingSpeakAddress, 80))
124.        {
125.          client.print("POST /update HTTP/1.1\n");
126.          client.print("Host: api.thingspeak.com\n");
127.          client.print("Connection: close\n");
128.          client.print("X-THINGSPEAKAPIKEY: "+writeAPIKey+"\n");
129.          client.print("Content-Type: application/x-www-form-
   urlencoded\n");
130.          client.print("Content-Length: ");
131.          client.print(tsData.length());
132.          client.print("\n\n");
133.
134.          // Envio da string para o Thingspeak
135.          client.print(tsData);
136.
137.          // Atribuicao do tempo a variavel lastConnectionTime para calculo
    de atualizacao
138.          lastConnectionTime = millis();
139.
140.          //Condicional de conexao ao Thingspeak
141.          if (client.connected())
142.          {
143.            // Debug de conectado ao Thingspeak
144.            Serial.println("Connecting to ThingSpeak...");
145.            Serial.println();
146.
147.            // Zera o contador de erro de conexao
148.            failedCounter = 0;
```

```
149.          }
150.          // Caso cliente nao tenha conectado corretamente
151.          else
152.          {
153.            // Incrementa o contador de falha
154.            failedCounter++;
155.
156.            // Print no Serial Monitor do número de tentativas de conexao r
   ealizadas
157.            Serial.println("Connection to ThingSpeak failed ("+String(faile
   dCounter, DEC)+")");
158.            Serial.println();
159.          }
160.
161.        }
162.
163.        // Se o cliente nãoestiver conectado no endereco da api, na porta l
   ogica 80, executa o condicional abaixo
164.        else
165.        {
166.          // Incrementa o contador de falha
167.          failedCounter++;
168.
169.          // Print no Serial Monitor do número de tentativas de conexao rea
   lizadas
170.          Serial.println("Connection to ThingSpeak Failed ("+String(failedC
   ounter, DEC)+")");
171.          Serial.println();
172.
173.          // Atribuicao do tempo na variavel lastConnectionTime para calcul
   o de atualizacao
174.          lastConnectionTime = millis();
175.        }
176.      } // Termino do metodo updateThingSpeak
177.
178.      // Inicio do metodo startEthernet
179.      void startEthernet()
180.      {
181.        // Parada do cliente, para garantir que "comeca do zero"
182.        client.stop();
183.
184.        Serial.println("Connecting Arduino to network...");
185.        Serial.println();
186.
187.        delay(1000); // Atraso de 1s
188.
189.        // Conexao na rede e atribuicao de endereco IP atraves de DHCP
190.        // Se o MAC Address for zero, nao executa a distribuicao de enderec
   o
191.        if (Ethernet.begin(mac) == 0)
192.        {
193.          Serial.println("DHCP Failed, reset Arduino to try again");
194.          Serial.println();
195.        }
196.        // Caso o MAC Address nao seja nulo, ou seja, tenha um conjunto val
   ido de valores
197.        else
198.        {
199.          Serial.println("Arduino connected to network using DHCP");
```

```
200.          Serial.println();
201.        }
202.
203.        delay(1000); // Atraso de 1s
204.      } // Termino do metodo startEthernet
205.
206.      // Inicio do metodo critical
207.      int critical(int atual, int antigo)
208.      {
209.        int critc = 5;
210.        if (atual == 0)
211.        {
212.          atual = 1;
213.        }
214.        if (antigo == 0)
215.        {
216.          antigo = 1;
217.        }
218.        double razao = (double) antigo / (double) atual;
219.
220.        // Checa para ver se a razao entre o valor atual e o valor antigo e
    ' menor que 1 (regredindo)
221.        if (razao < 1 )
222.        {
223.          if (atual <= 15)
224.          {
225.            critc = 3;
226.          }
227.          else if (atual > 15 && atual <= 30)
228.          {
229.            critc = 4;
230.          }
231.          else if (atual > 30 && atual <= 40)
232.          {
233.            critc = 5;
234.          }
235.          else
236.          {
237.            critc = 5;
238.          }
239.        }
240.        // Checa para ver se o valor da razao esta' entre 1 e 1.1 (aumentou
    nivel no maximo 10%)
241.        else if (razao >= 1 &&  razao < 1.1)
242.        {
243.          if (atual <= 15)
244.          {
245.            critc = 3;
246.          }
247.          else if (atual > 15 && atual <= 30)
248.          {
249.            critc = 4;
250.          }
251.          else if (atual > 30 && atual <= 40)
252.          {
253.            critc = 4;
254.          }
255.          else
256.          {
```

```
257.            critc = 5;
258.          }
259.        }
260.        // Checa para ver se o valor da razao esta' entre 1.1 e 1.2 (aument
    ou nivel no maximo 20%)
261.        else if (razao >= 1.1 &&  razao < 1.2)
262.        {
263.          if (atual <= 15)
264.          {
265.            critc = 2;
266.          }
267.          else if (atual > 15 && atual <= 30)
268.          {
269.            critc = 3;
270.          }
271.          else if (atual > 30 && atual <= 40)
272.          {
273.            critc = 4;
274.          }
275.          else
276.          {
277.            critc = 5;
278.          }
279.        }
280.        // Checa para ver se o valor da razao esta' entre 1.2 e 1.3 (aument
    ou nivel no maximo 30%)
281.        else if (razao >= 1.2 &&  razao < 1.3)
282.        {
283.          if (atual <= 15)
284.          {
285.            critc = 2;
286.          }
287.          else if (atual > 15 && atual <= 30)
288.          {
289.            critc = 2;
290.          }
291.          else if (atual > 30 && atual <= 40)
292.          {
293.            critc = 3;
294.          }
295.          else
296.          {
297.            critc = 4;
298.          }
299.        }
300.        // Checa para ver se o valor da razao e' maior do que 1.3 (aumentou
     nivel acima de 30%)
301.        else if (razao >= 1.3)
302.        {
303.          if (atual <= 15)
304.          {
305.            critc = 1;
306.          }
307.          else if (atual > 15 && atual <= 30)
308.          {
309.            critc = 2;
310.          }
311.          else if (atual > 30 && atual <= 40)
312.          {
```

```
313.             critc = 2;
314.           }
315.           else
316.           {
317.             critc = 3;
318.           }
319.         }
320.
321.         return critc;
322.       } // Termino do metodo critical
```

9 Anexo B

9.1 Código do modulo Router

```
1. #include <NewPing.h> //biblioteca para utilizar o sonar
2.
3. //defines sonar
4. #define ECHO_PIN 11 // Arduino pin tied to echo pin on the ultrasonic senso
   r.
5. #define TRIGGER_PIN 12 // Arduino pin tied to trigger pin on the ultrasonic
    sensor.
6. #define MAX_DISTANCE 500 // Maximum distance we want to ping for (in centim
   eters). Maximum sensor distance is rated at 400-500cm.
7.
8. NewPing sonar(TRIGGER_PIN, ECHO_PIN, MAX_DISTANCE); // NewPing setup of pin
   s and maximum distance.
9.
10. void setup()
11. {
12.   Serial.begin(9600);
13. }
14. void loop()
15. {
16.   unsigned int uS = sonar.ping(); // Send ping, get ping time in microsecon
   ds (uS).
17.   unsigned int cm = sonar.convert_cm(uS); // Convert into centimeters
18.   delay(10000);
19.   Serial.println(String(cm));
20. }
```