



FACULDADE DE TECNOLOGIA SENAI CIMATEC
ESPECIALIZAÇÃO EM AUTOMAÇÃO, CONTROLE E ROBÓTICA INDUSTRIAL

ANDRÉ PINTO MAROTTA

**UTILIZAÇÃO DE SISTEMAS *OPEN SOURCE* NO MONITORAMENTO DE
INSUMOS INDUSTRIAIS**

Salvador

2015

ANDRÉ PINTO MAROTTA

**UTILIZAÇÃO DE SISTEMAS OPEN SOURCE NO MONITORAMENTO DE
INSUMOS INDUSTRIAIS**

Trabalho de conclusão de curso apresentada à Faculdade de Tecnologia SENAI CIMATEC como requisito final para a obtenção do Título de Especialista em Automação, Controle e Robótica Industrial.

Orientador: Prof. M.Sc. Cléber Vinícius Ribeiro de Almeida.

Salvador
2015

Ficha catalográfica elaborada pela Biblioteca da Faculdade de Tecnologia SENAI CIMATEC

M355u Marotta, André Pinto

Utilização de sistemas Open Source no monitoramento de insumos industriais
/ André Pinto Marotta. – Salvador, 2015.

85 f. : il. color.

Orientador: Prof. MSc. Cléber Vinícius Ribeiro de Almeida.
Coorientador: Prof. MSc. Oberdan Rocha Pinheiro.

Monografia (Especialização em Automação, Controle e Robótica) – Programa
de Pós-Graduação, Faculdade de Tecnologia SENAI CIMATEC, Salvador, 2015.
Inclui referências.

1. Gerenciamento de insumos industriais. 2. Open Source. 3. Ethernet. 4.
Arduino Mega. I. Faculdade de Tecnologia SENAI CIMATEC. II. Almeida, Cléber
Vinícius Ribeiro de. III. Pinheiro, Oberdan Rocha. IV. Título.

CDD 629.8

ANDRÉ PINTO MAROTTA

**UTILIZAÇÃO DE SISTEMAS *OPEN SOURCE* NO MONITORAMENTO DE
INSUMOS INDUSTRIAIS**

Projeto Final de Curso aprovado com nota 9,0 (nove) como requisito de Especialista em Automação, Controle e Robótica, tendo sido julgado pela Banca Examinadora formada pelos Professores:

Msc. Cléber Vinícius Ribeiro de Almeida – Orientador

Faculdade de Tecnologia SENAI CIMATEC

Msc. Oberdan Rocha Pinheiro – Professor

Faculdade de Tecnologia SENAI CIMATEC

Salvador, 15 de outubro de 2015

Dedico este Trabalho a meus familiares, amigos e professores, os quais,
cada um, da sua forma, foram fundamentais para a finalização deste
Trabalho.

AGRADECIMENTOS

Agradeço a Deus, nosso pai maior por tudo;

Aos meus pais, que me ensinaram a ser uma pessoa honesta e a acreditar;

À minha amada, por tornar minha vida tão feliz;

A todos os amigos que fiz ao longo deste curso, em especial a Daniel, Fernando e Iuri, aos quais me ensinaram muito;

À instituição e a todos os professores do curso de pós-graduação que contribuíram diretamente para a minha formação;

Em especial ao professor Cléber Vinícius pela dedicação e paciência que foram fundamentais para a conclusão deste trabalho;

Em especial também ao professor Oberdan pela presteza, amizade e toda a boa vontade e disponibilidade em me ajudar sempre que precisei.

“Vamos inventar o amanhã e parar de nos preocupar com o passado”.

Steve Jobs

RESUMO

Este trabalho apresenta uma proposta de ferramenta alternativa para gerenciamento do consumo de insumos. Compõe-se de um projeto e de uma implementação de um sistema de aquisição de dados para medir nível e vazão de insumos de processo fluido até a construção de um protótipo para validação conceitual, utilizando dispositivos *Open Source*. O sistema proposto é baseado no microcontrolador ATMEGA 2560 (*Arduino* MEGA) e utiliza uma placa *Ethernet* para enviar os dados coletados através de uma rede de comunicação aberta (*Ethernet* e Modbus-TCP), viabilizando o armazenamento das variações em uma base de dados gerenciada por um *software* de supervisão e aquisição de dados desenvolvido para esta aplicação. Posteriormente, são analisados os resultados obtidos e apresentadas as considerações finais.

Palavras-chave: Gerenciamento de Insumos Industriais, *Open Source*, *Ethernet*, *Arduino* Mega.

ABSTRACT

This paper presents a proposal for an alternative tool for managing the consumption of raw materials. It consists of a design and an implementation of a data acquisition system to measure level and flow of fluid process of raw materials up to the construction of a prototype for concept validation using Open Source devices. The proposed system is based on microcontroller ATMEGA 2560 (Arduino MEGA) and it uses an Ethernet card to send the data collected through an open communication network (Ethernet and Modbus-TCP), enabling the variations storage in a database managed by supervisory software and a data acquisition developed for this application. The next section discusses the results obtained and presented the closing remarks.

Keywords: Management of Industrial Inputs, Open Source, Ethernet, Arduino Mega.

LISTA DE FIGURAS

Figura 2.1: Visor de nível	18
Figura 2.2: Flutuador	19
Figura 2.3: Sensor de nível por pressão diferencial.....	20
Figura 2.4: Sensor de nível por ultrassom	21
Figura 2.5: Sensor de nível por radar.....	22
Figura 2.6: Medição de vazão por placa de orifício.....	24
Figura 2.7: Medição de vazão por tubo de Venturi	24
Figura 2.8: Medição de vazão por tubo de Pitot.....	25
Figura 2.9: Medição de vazão por turbina.....	25
Figura 2.10: Pinagem do <i>Arduino Mega 2560</i>	27
Figura 2.11: <i>Ethernet Shield</i>	27
Figura 2.12: <i>Data Shield</i>	28
Figura 2.13: Comunicação SPI	29
Figura 2.14: Comunicação I2C.....	29
Figura 2.15: Bits de comunicação I2C	30
Figura 2.16: Frames Modbus-RTU / ASC II	32
Figura 2.17: Frame Modbus-TCP.....	32
Figura 3.1: Diagrama de funcionamento do sistema proposto.....	34
Figura 3.2: Diagrama de montagem do <i>hardware</i>	35
Figura 3.3: Disposição de montagem do <i>hardware</i>	35
Figura 3.4: Display serial 16x2	36
Figura 3.5: HC-RS04 e YF-S201: Sensores de nível e vazão	36
Figura 3.6: Arduino e <i>Ethernet shield</i>	37
Figura 3.7: <i>Data Shield</i>	37

Figura 3.8: Diagrama do projeto de comunicação do sistema proposto	38
Figura 3.9: Diagrama do projeto do software proposto	39
Figura 4.1: Protótipo do sistema proposto	40
Figura 4.2: Validação da leitura do sensor de nível	41
Figura 4.3: Validação da leitura do sensor de vazão	41
Figura 4.4: Validação da comunicação UDP e Modbus-TCP	42
Figura 4.5: Curva de erro versus graduação.....	44
Figura 4.6: Curva de vazão versus frequência	44
Figura 4.7: Curva de desvio do erro médio em função da frequência	45
Figura 4.8: Validação da comunicação UDP.....	46
Figura 4.9: Validação da comunicação Modbus-TCP	46
Figura 4.10: Validação da leitura da base de dados	47
Figura 4.11: Validação da leitura da memória de massa	47

LISTA DE TABELAS

Tabela 4.1: Variação de nível.....	43
Tabela 4.2: Dados de vazão e frequência.....	45
Tabela 4.3: Comparativo de custos entre equipamentos.....	48

LISTA DE ABREVIATURAS E SIGLAS

A/D	<i>Analog to digital</i> (Analógico para Digital)
CPU	<i>Central Processing Unit</i> (Unidade de Processamento Central)
D/A	<i>Digital to Analog</i> (Digital para Analógico)
I/O	<i>Input / Output</i> (Entrada / Saída)
I2C	<i>Inter-Integrated Circuit</i> (Circuito Inter-Integrado)
ICSP	<i>In Chip System Programming</i> (Programação do Sistema em <i>Chip</i>)
IP	<i>Internet Protocol</i> (Protocolo de <i>Internet</i>)
MISO	<i>Master In to Slave Out</i> (Entrada do Mestre para Saída do Escravo)
MOSI	<i>Master Out to Slave In</i> (Saída do Mestre para Entrada do Escravo)
PWM	<i>Pulse Width Modulation</i> (Modulação por Largura de Pulso)
SPI	<i>Serial Peripheral Interface</i> (Interface Serial do Periférico)
TCP	<i>Transmission Control Protocol</i> (Protocolo de Controle de Transmissão)
UART	<i>Universal Asynchronous Receiver/Transmitter</i> (Receptor/Transmissor Assíncrono Universal)
UDP	<i>User Datagram Protocol</i> (Protocolo de Datagrama do Usuário)
USB	<i>Universal Serial Bus</i> (Barramento Serial Universal)

SUMÁRIO

1 INTRODUÇÃO	15
1.1 CONTEXTUALIZAÇÃO.....	15
1.2 OBJETIVOS.....	15
1.3 ORGANIZAÇÃO DO TEXTO	16
2 FUNDAMENTOS.....	17
2.1 MEDIDORES DE NÍVEL LÍQUIDOS	18
2.1.1 Medição Direta	18
2.1.1.1 Visores	18
2.1.1.2 Flutuadores	19
2.1.2 Medição Indireta	19
2.1.2.1 Medição por pressão diferencial	20
2.1.2.2 Medição por ultrassom	21
2.1.2.3 Medição por radar.....	22
2.2 MEDIDORES DE VAZÃO	23
2.2.1 Placa de Orifício	24
2.2.2 Tubo de Venturi	24
2.2.3 Tubo de Pitot	25
2.2.4 Turbina	25
2.3 ELEMENTOS PARA, AQUISIÇÃO DE DADOS E COMUNICAÇÃO.....	26
2.3.1 Arduino Mega	26
2.3.2 Ethernet Shield	27
2.3.3 Data Shield.....	27
2.3.4 Protocolos SPI, ICSP e I2C	28
2.3.5 Protocolos TCP e UDP.....	30
2.3.6 Protocolo Modbus-TCP.....	31
3 PROJETO E IMPLEMENTAÇÃO.....	34
3.1 ABORDAGEM DO SISTEMA PROPOSTO.....	34

3.2 <i>HARDWARE</i>	35
3.3 <i>COMUNICAÇÃO</i>	38
3.4 <i>SOFTWARE</i>	39
4 VALIDAÇÃO EXPERIMENTAL	40
4.1 MÉTODO DA EXPERIMENTAÇÃO	40
4.2 INFRA-ESTRUTURA DE TESTES	40
4.3 RESULTADOS DOS TESTES	43
4.4 AVALIAÇÃO DOS RESULTADOS	48
5 CONCLUSÃO	50
ANEXOS	51
ANEXO A: ESQUEMÁTICO DO <i>DATA SHIELD</i>	51
ANEXO B: ESQUEMÁTICO DO <i>ETHERNET SHIELD</i> REV. 03.....	52
ANEXO C: DATASHEET DO ATMEGA 2560.....	53
APÊNDICE	55
APÊNDICE A: CÓDIGO FONTE DO ATMEGA 2560, <i>ARDUINO</i> IDE 1.0.5.....	55
REFERÊNCIAS.....	82

1 INTRODUÇÃO

O gerenciamento de insumos (energia, água, gás, vapor, óleo diesel, ácidos e etc.) é uma forma de monitorar um sistema através de dois pontos de vista: o técnico e o gerencial. Conforme (BARROS, 2010), do ponto de vista técnico visa avaliar desperdícios, vazamentos e verificar se as máquinas e processos operam em condições normais em termos de consumo, grandezas e rendimento. Conforme (ALBERTO, 2014) do ponto de vista gerencial busca avaliar custos e avaliar continuamente a sua influência no comparativo ao produto final sobretudo na relação por tonelada produzida.

1.1 CONTEXTUALIZAÇÃO

A partir dessa análise, abolir o antigo método de o leiturista mensalmente verificar totalizadores locais e passar a utilizar a automação e as redes de comunicação para obter estas informações em tempo real e visualizá-las através do conforto de um terminal corporativo, potencializam a competitividade conforme apresentado em (CNI, 2013), sobretudo na velocidade de decisão gerencial e na gestão dos custos da instalação. Deste ponto em diante, uma série de tecnologias e sistemas de automação podem ajudar nesta problemática, entretanto sempre atreladas a um alto custo de implantação e a sistemas com tecnologia proprietária (TETRA, 2011).

1.2 OBJETIVO

Em contraposição a este cenário, este documento visa apresentar uma contribuição para a redução de importações de equipamentos específicos em detrimento do projeto e montagem de soluções alternativas *Open Source* de baixo custo disponíveis no mercado. Através da utilização de tecnologias abertas (OSI,1998) e demais dispositivos de tecnologia aberta, como o microcontrolador ATMEGA 2560 (ARDUINO,2014) (MONK,2014) e os protocolos *Ethernet* e Modbus-TCP (MODBUS,2014), este Trabalho pretende apresentar um modelo e confeccionar um protótipo para medição de nível e vazão em menor escala para validação conceitual.

1.3 ORGANIZAÇÃO DO TEXTO

No Capítulo 2, estão expostos os fundamentos do estado da arte no entendimento da instrumentação (WILLIAN.C,2003) especificamente sistemas de medição de vazão e nível (WEBSTER, 1999), (MORSE I.E.,1989), apresentados a fim de se avaliar os tipos de sistemas possíveis de utilização em um modelo que contemple a medição de vazão e nível. Dentre as bases fundamentais ao desenvolvimento do trabalho, percorridas no Capítulo 2, tem-se: sistemas abertos (OSI,1998); (ARDUINO, 2014) armazenamento de dados em memória de massa através de dispositivos eletrônicos (IDOETA,2008), dispostos com o intuito de apresentar os recursos de *hardware* necessários para desenvolvimento de um sistema ou um protótipo; assim como os protocolos de comunicação abertos (OSRWA, 2014) (MODBUS. 2014) que possibilitam troca de dados entre um sistema eletrônico e um computador.

Após a definição dos constructos e de um vocabulário comum ao leitor realizada no Capítulo 2, no Capítulo 3 está descrito o projeto do sistema de gerenciamento proposto. O modelo utiliza recursos de *hardware* e sensores de vazão e nível selecionados dentre os descritos no Capítulo 2.

No Capítulo 4 está descrita a apresentação dos testes, da metodologia e da infraestrutura utilizada para obtenção de valores de medição de vazão e nível, além dos testes de comunicação, validação e análise dos resultados experimentais obtidos com este sistema. Por fim, o Capítulo 5, apresenta a conclusão e as considerações finais acerca do presente Trabalho realizado.

2 FUNDAMENTOS

A evolução natural da área de instrumentação com o advento dos transdutores eletrônicos, os quais traduziram variações de grandezas em variações elétricas padronizadas e sua posterior transmissão em padrões analógicos, permitiu que estas informações fossem transmitidas do campo para as conhecidas salas de controle (DUNN, 2013).

Atualmente, os protocolos de comunicação e as redes, tem papel fundamental neste processo de aquisição e transmissão (LUGLI, 2013). Essa conjunção entre instrumentos de campo eletrônico, redes de comunicação, sistemas de supervisão e armazenamento de dados em um computador, crua um cenário favorável em termos de automação para que se utilizem estes dados de medição para funções que estão além do controle do processo: o gerenciamento (BARROS, 2010).

A universalização deste tipo de conceito, prescinde de uma forma de organização destes agentes de automação, que consigam atender necessidades para instalações menores e que necessitem de um custo menor para garantir sua viabilidade (TETRA, 2011). A proposta para utilização de sistemas abertos vem ganhando força perante a comunidade acadêmica e o mercado, como formas de universalizar o acesso ao núcleo das tecnologias comumente utilizadas como verificado em (SERPRO, 2015).

2.1 MEDIDORES DE NÍVEL LÍQUIDOS

Os medidores de nível líquido apresentados em (WEBSTER,1999) são instrumentos que tem como objetivo medir a altura de colunas de líquidos com base em um ponto de referência para se estimar a quantidade deste fluido. Podem ser classificados pelo tipo de medição como: medição direta e medição indireta.

2.1.1 MEDIÇÃO DIRETA

Medição direta é o tipo de medição pelo qual se obtém o valor experimental diretamente segundo as unidades de medidas respectivas. Para medição de nível os tipos mais comuns são: Visores e Flutuadores.

2.1.1.1 VISORES

Neste tipo de instrumento é instalado um tubo transparente com régua com graduação desde a base do reservatório até ao topo do mesmo se este for fechado ou com abertura para atmosfera se for aberto, como mostra a Figura 2.1.

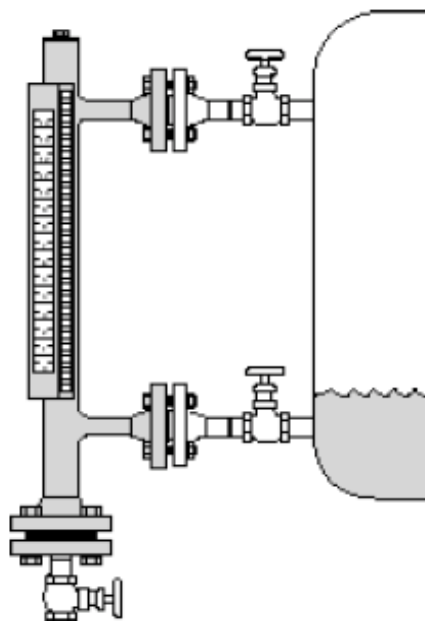


Figura 2.1 – Visor de nível

Fonte: EBAH, 2015.

2.1.1.2 FLUTUADORES

Neste tipo de instrumento é confeccionado um elemento mecânico que transforme a variação vertical da superfície do líquido em informação direta de nível. Exemplo é ilustrado pela Figura 2.2.

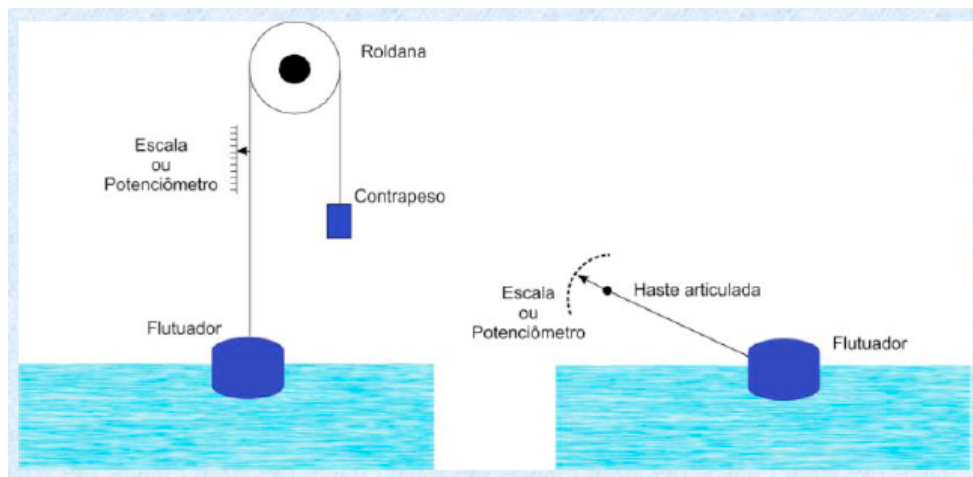


Figura 2.2 – Flutuador
Fonte: OLIVEIRA.A, 2015.

2.1.2 MEDIÇÃO INDIRETA

Medição indireta é Medição direta é o tipo de medição pelo qual se obtém o valor experimental através de relações entre a grandeza que se deseja obter e outras grandezas distintas. Para medição de nível os tipos mais comuns são:

- Pressão diferencial;
- Ultrassom;
- Radar.

2.1.2.1 MEDIÇÃO POR PRESSÃO DIFERENCIAL

Neste método de medição, o instrumento geralmente baseado em capacitor diferencial (WEBSTER,1999) é utilizado para medição de reservatórios abertos ou fechados. Nestes casos, sua instalação obedece ao princípio similar ao do visor de nível, aonde a parte inferior do instrumento necessita de contato com o fluido. A Figura 2.3 ilustra o exemplo.

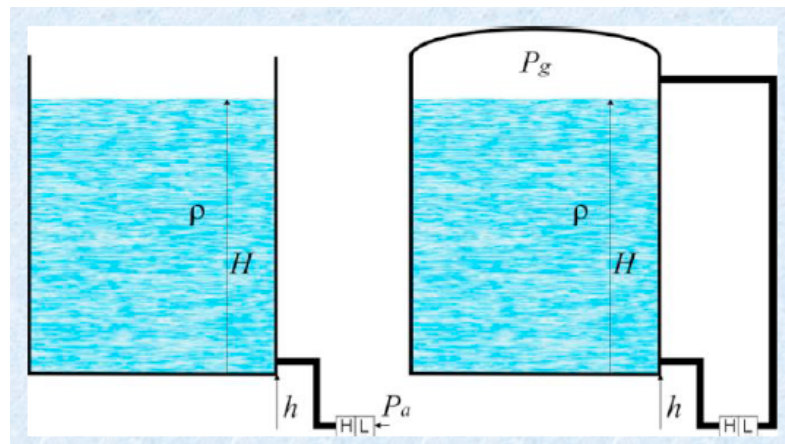


Figura 2.3 – Sensor de nível por pressão diferencial

Fonte: OLIVEIRA,A.,2015.

Conforme a Figura 2.3, para o reservatório aberto:

$$P_h = \rho G(H + h) + P_a \quad (2.1)$$

$$P_l = P_a \quad (2.2)$$

Para o reservatório fechado:

$$P_h = \rho G(H + h) + P_g \quad (2.3)$$

$$P_l = P_g \quad (2.4)$$

Para ambos os casos:

$$P_h - P_l = \rho G(H + h) \quad (2.5)$$

2.1.2.2 MEDIÇÃO POR ULTRASSOM

Neste método de medição, o sensor fica posicionado no topo do reservatório com a face virada diretamente para o fluido. O sensor emite uma onda sonora de frequência inaudível e quando esta onda entra em contato com a superfície do fluido, parte desta onda é refletida de volta (CAETANO.A, 2013). O valor do nível é obtido pelo cálculo do tempo TC que a onda demora entre ser emitida, refletida e retornar para o equipamento. A Figura 2.4 ilustra esta situação.

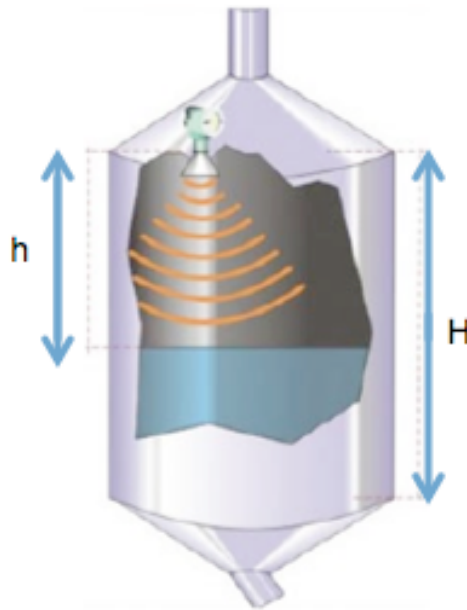


Figura 2.4 – Sensor de nível por ultrassom.

Fonte: INSTRUMATIC, 2015.

Conforme a Figura 2.4, para o reservatório:

$$2(H - h) = TC \quad (2.6)$$

Onde:

- Velocidade = distância X tempo.
- Velocidade do som = 331,4 m/s
- Distância = Velocidade / Tempo

2.1.2.3 MEDIÇÃO POR RADAR

Neste método o funcionamento é similar ao da medição por ultrassom, entretanto a diferença é ao invés de ser emitida uma onda mecânica sonora, é emitida uma onda eletromagnética (MORSE I.E.,1989). A sua exatidão é muito maior que a do ultrassom, mas seu custo é muito mais alto. A Figura 2.5 ilustra seu funcionamento.

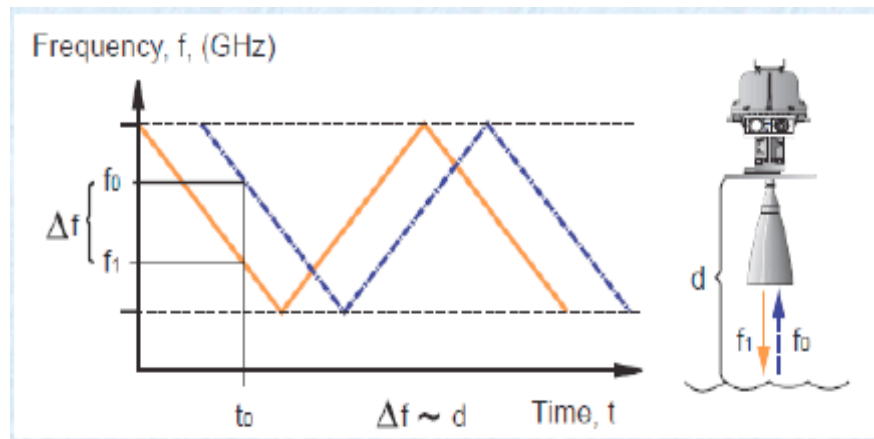


Figura 2.5 – Sensor de nível por radar.

Fonte: CAETANO.A.,2015.

Conforme a Figura 2.5, para o reservatório têm-se:

$$df = -\frac{df_1}{dt} t = \frac{df_1}{dt} \frac{2d}{c} \quad (2.7)$$

Portanto, de (2.7) obtém-se (2.8):

$$d = -\frac{df_c dt}{2df_1} \quad (2.8)$$

2.2 MEDIDORES DE VAZÃO

Os medidores de vazão de fluido apresentados em (WEBSTER,1999) são instrumentos que tem como objetivo medir a velocidade de escoamento de um fluido através de um duto de características conhecidas, os quais obedecem ao princípio da equação de Bernouilli (STREET, 1969), (HALLYDAY, 2003) conforme (2.9).

$$\frac{P_1}{\rho} + \frac{(V_1)^2}{2} + gz_1 = P_2 + \frac{(V_2)^2}{2} + gz_2 \quad (2.9)$$

Uma técnica comumente utilizada para obter vazão de fluidos é através da diferença de pressão entre dois pontos (OLIVEIRA. A, 2015), modelada por (2.10).

$$Q = \frac{A_2}{[1 - (\frac{A_2}{A_1})^2]^{0.5}} \left[\frac{2(P_1 - P_2)}{\rho} \right]^{0.5} \quad (2.10)$$

Os tipos de medidores mais conhecidos utilizam os métodos:

- Medição por placa de orifício;
- Medição por tubo de Venturi;
- Medição por tubo de Pitot;
- Medição por turbina.

2.2.1 PLACA DE ORIFÍCIO

Neste método de medição, o instrumento obtém a vazão através da medição da diferença de pressão causada pela redução brusca de diâmetro como consequência da placa de orifício instalada em um duto (MORSE I.E.,1989). A Figura 2.6 exemplifica.

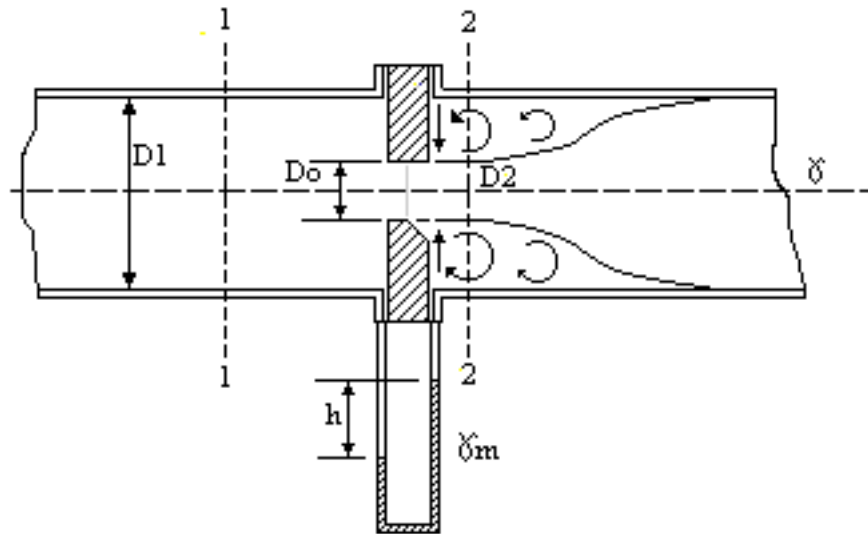


Figura 2.6 – Medição de vazão por placa de orifício.

Fonte: WIKIBOOKS, 2015.

2.2.2 TUBO DE VENTURI

Neste método de medição, a curvatura ao longo do tubo que possui diâmetros diferentes ao longo da sua estrutura, faz com que haja uma diferença de pressão entre o ponto de entrada com diâmetro maior e o ponto intermediário com diâmetro menor, possibilitando a medição da vazão através da diferença de pressão entre estes dois pontos (MORSE I.E., 1989) como apresenta a Figura 2.7.



Figura 2.7– Medição de vazão por tubo de Venturi.

Fonte: ITUFLUX, 2015.

2.2.3 TUBO DE PITOT

Neste método de medição, o pino posicionado na parte externa do duto, exerce uma pressão vertical para cima de forma proporcional à pressão sentida em sua outra extremidade, oriunda do deslocamento do fluido que passa por este duto (MORSE I.E.,1989). O sensor então pode obter o valor de vazão através desta diferença de pressão. Situação ilustrada na Figura 2.8.

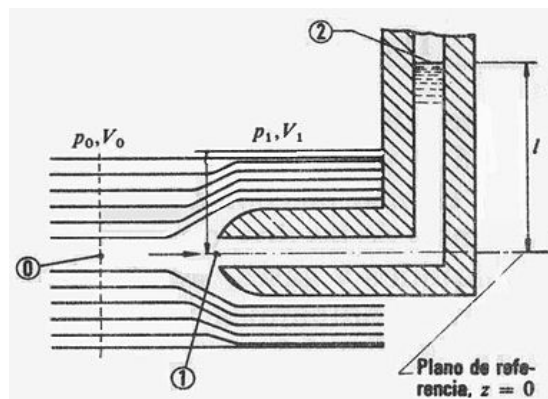


Figura 2.8 – Medição de vazão por tubo de Pitot.

Fonte: OLIVEIRA.A., 2015.

2.2.4 TURBINA

Neste método de medição, existe uma turbina conectada em série com o duto. O fluido que passa pelo duto faz a turbina girar com uma velocidade proporcional à velocidade do deslocamento do fluido (INCONTROL, 2015). Para isto pode ser utilizado um sensor de Efeito Hall ou foto-acoplador. Este método está ilustrado na Figura 2.9.



Figura 2.9 – Medição de vazão por turbina

Fonte: INCONTROL, 2015.

2.3 ELEMENTOS PARA AQUISIÇÃO DE DADOS E COMUNICAÇÃO

Microcontroladores são sistemas computacionais que possuem unidade central de processamento (CPU) memórias de dados e programa, *clock* e uma série de periféricos e acessórios como entradas e saídas (I/O), conversores de sinais analógicos para digitais (conversores A/D) e de sinais digitais para analógicos (conversores D/A) e interfaces de comunicação em um mesmo componente, o qual pode ser programado para execução de funções específicas (NERYS, 2010).

2.3.1 ARDUINO MEGA

O *Arduino Mega* é uma placa eletrônica de interface aberta baseada no microcontrolador ATMEGA 2560. O *Arduino* e seus “clones” (versões das cópias de fabricantes diferentes, uma vez que se trata de um *hardware* de código aberto) fazem uso do conceito de “*Shields*” (escudos, em inglês), os quais são placas de circuito impresso, normalmente fixadas no topo do aparelho através de uma conexão alimentada por pinos-conectores, designadas para serem expansões da placa principal para disponibilização de várias funções específicas, desde a manipulação de motores até sistemas de rede sem fio (WIKIPEDIA, 2015).

A placa possui 54 pinos de I/O digital, dos quais 15 podem ser programados como Modulação por Largura de Pulso (*Pulse Width Modulation* – PWM), 16 entradas analógicas, 4 portas seriais de *hardware* do tipo UART (acrônimo inglês para “*Universal Asynchronous Receiver/Transmitter*”, ou “Receptor/Transmissor Assíncrono Universal”), cristal oscilador de 16 MHz, conexão USB (*Universal Serial Bus*), conector para fonte de alimentação e leitor ICSP, além de possuir 256 KB de memória para programação, das quais 8 KB são utilizadas pelo código *bootloader*, uma parte independente do *firmware* responsável por inicializar o sistema (*hardware*) e deixá-lo pronto para a substituição de novos *firmwares* (ARDUINO, 2014).

A Figura 2.10 obtida em (ARDUINO,2014) mostra o diagrama do encapsulamento do Arduino MEGA.

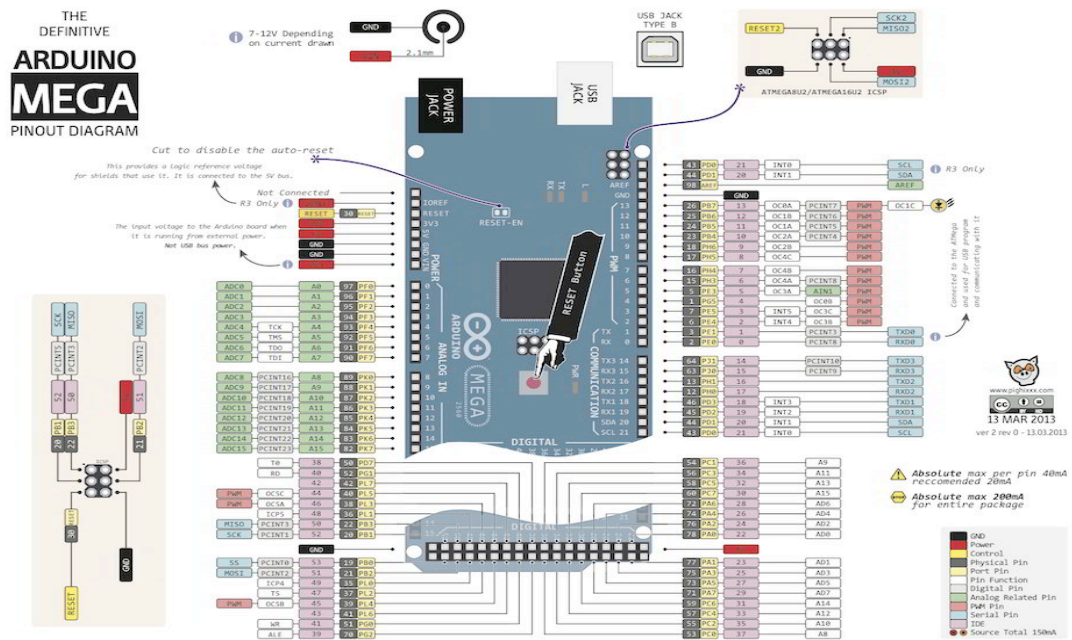


Figura 2.10 – Pinagem do Arduino MEGA 2560.

Fonte: ARDUINO, 2014

2.3.2 ETHERNET SHIELD

O *Ethernet Shield* é uma placa de circuito impresso que possui configuração de pinos (“pinagem”) compatível com o *Arduino*. Sua função principal é viabilizar a comunicação *Ethernet* para o *Arduino* (ARDUINO,2014). A Figura 2.11 apresenta o equipamento.

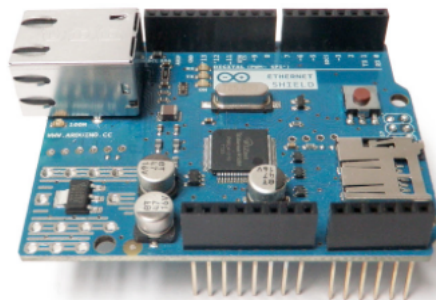


Figura 2.11 – *Ethernet Shield*.

Fonte: ARDUINO, 2014.

2.3.3 DATA SHIELD

O *Data Shield* é uma placa de circuito impresso desenvolvida por (LABORATORIO DE GARAGEM, 2015). Também possui “pinagem” compatível com o *Arduino*. Sua

função é adicionar os circuitos relógio de tempo real, Display, botões, bateria e cartão de memória ao *Arduino*, para viabilizar a aquisição e armazenamento de dados em cartão de memória, com a data e hora programadas. A Figura 2.12 apresenta o dispositivo.

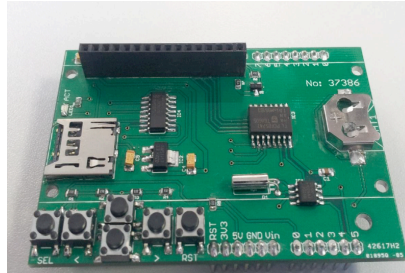


Figura 2.12 – *Data Shield*.

Fonte: LABORATORIO DE GARAGEM,2015.

1.3.4 PROTOCOLOS SPI, ICSP e I2C

A comunicação *Serial Peripheral Interface* – Interface Serial Periférica (SPI) (MACIEL, 2015) é um protocolo de comunicação do tipo mestre/escravo que permite a comunicação serial rápida entre dois ou mais chips. Existem duas linhas de comunicação: do mestre para os escravos *Master In to Slave Out* – Saída do escravo para o mestre (MISO) e do escravo para o mestre *Master Out to Slave In* – Saída do mestre para o escravo (MOSI).

Além destes dois caminhos, existe a função *Chip Select* – Seleção de Chip (SS) o qual o mestre pode habilitar o pino do circuito integrado o qual deseja enviar ou receber informações. Há também a necessidade de pulsos de *clock* em SCLK para sincronizar a troca de dados, uma vez que a comunicação é síncrona e os dispositivos estão sempre enviando e recebendo dados ao mesmo tempo (MACIEL, 2015). A Figura 2.13 ilustra exemplo de ligação de componentes em SPI.

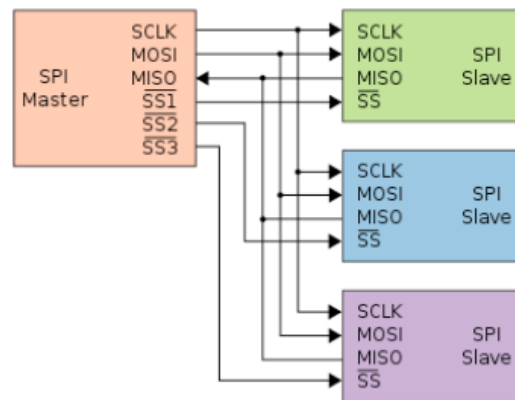


Figura 2.13 – Comunicação SPI.

Fonte: CAETANO,A., 2015.

A comunicação *In Circuit Serial Programming* – Programação serial no circuito (ICSP) obedece ao mesmo princípio da comunicação SPI, entretanto tem o objetivo de realizar a comunicação USB entre o microcontrolador e o computador, para gravação de códigos sem a necessidade de retirar o equipamento e colocar em um gravador (MACIEL. 2014).

A comunicação Circuitos Inter-Integrados, do acrônimo inglês para *Inter-Integrated Circuits* (I2C), é também uma comunicação serial entre dispositivos (*chips*). Entretanto, ela é assíncrona e permite apenas envio ou recebimento de dados, um de cada vez (MACIEL, 2015). O sistema permite a conexão de até 127 dispositivos. O Barramento I2C possui duas linhas. A Linha SCL, linha de *clock* e a linha SDA, linha de dados. Na Figura 2.14 está representado o barramento de comunicação I2C.

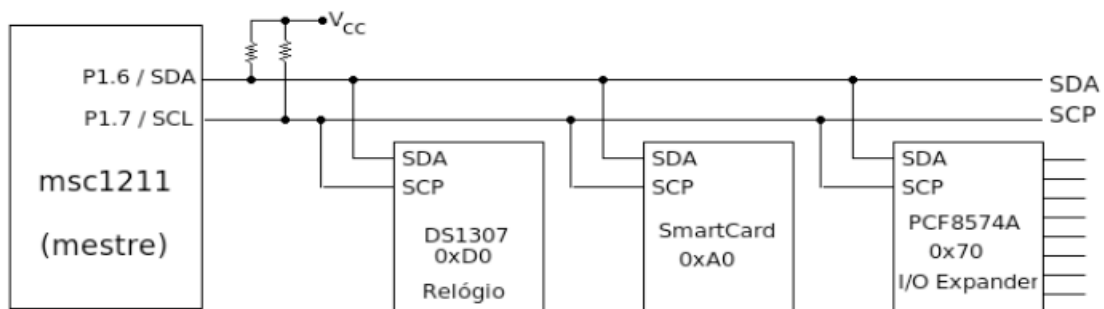


Figura 2.14 – Comunicação I2C.

Fonte: CAETANO.A, 2015.

Na comunicação I2C, só pode haver um mestre e um escravo se comunicando por vez. A Figura 2.15 apresenta a representação de uma linha de bits típica de uma comunicação I2C. O início e fim da transmissão ou início e fim da recepção, são sempre marcados pelos bits ACK e STOP.

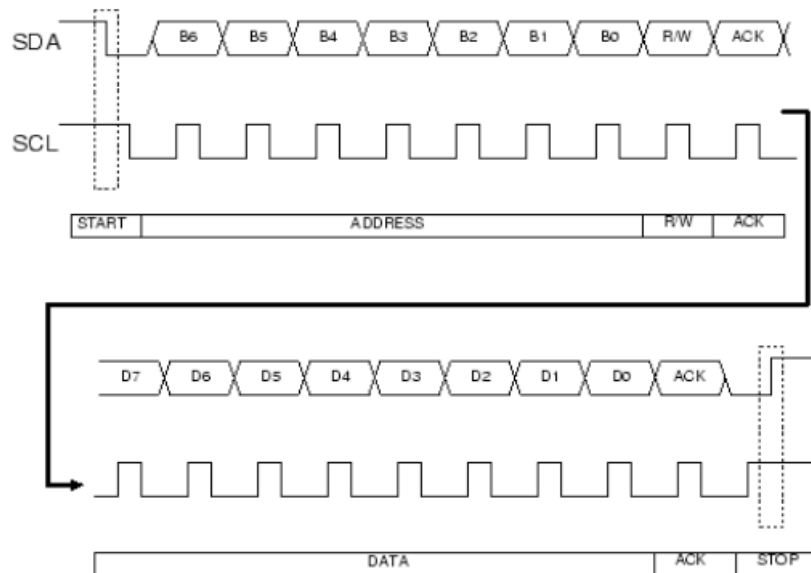


Figura 2.15 – Bits de Comunicação I2C.

Fonte: CAETANO.A., 2015.

2.3.5 PROTOCOLOS TCP e UDP

A comunicação Protocolo de Controle de Transmissão, do acrônimo inglês para *Transmission Control Protocol* (TCP), é um importante protocolo do conjunto de protocolos *Ethernet*. Ele está incluso no conjunto de protocolos que resultam no TCP/IP. Segundo (TCP/IP, 2014), suas funções principais são:

- Garantir a entrega de pacotes IP;
- Executar separação e reagrupamento de blocos de dados;
- Verificar a integridade dos dados transmitidos;
- Enviar mensagens de confirmação de recebimento de dados.

O TCP funciona com base em conexões. Para haver uma transação entre um dispositivo cliente e um dispositivo servidor, o primeiro envia um sinal SYN para o último que retorna com o sinal SYN combinado com o sinal ACK, dando início à comunicação. Então o primeiro responde com sinal ACK, habilitando a troca de dados. (TCP/IP, 2014).

A comunicação Protocolo de Utilização de Datagramas, do acrônimo inglês para *User Datagram Protocol* (UDP), é um protocolo similar ao TCP, porém com a diferença de que não é baseado em conexões. Significa que o mesmo possui menos segurança na transmissão dos dados, fazendo com que o protocolo UDP seja mais rápido que o TCP (TCP/IP, 2014).

Ambos protocolos têm sido utilizados, além da aplicação para comunicação entre computadores, também são amplamente utilizados para comunicação entre computadores e dispositivos de aquisição de dados, monitoramento e controle por exemplo.

2.3.6 PROTOCOLO MODBUS

A O protocolo de comunicação Modbus é um conjunto aberto de instruções desenvolvidas pela Modicon no final da década de 1970. Posteriormente a Modicon foi adquirida pelo grupo Schneider que passou em 2004 os direitos do protocolo para a *Modbus Organization*. O protocolo foi disponibilizado para toda a comunidade (MODBUS, 2014). Este protocolo tem sido usado em uma vasta gama de aplicações como:

- Instrumentos, multimedidores, transdutores e concentradores de dados;
- Automação residencial, comercial, naval e industrial.

O protocolo Modbus é bastante utilizado em virtude da facilidade de implementação. Utiliza padrões de meio físico:

- RS-232 (Máquina dispositivo – Ponto a ponto);
- RS-485 (Rede mestre-escravo – 1 a 247 endereços de equipamentos);
- *Ethernet* TCP/IP (Rede Ethernet - Modbus/TCP).

Os modos de transmissão especificados no protocolo Modbus são, Modbus-RTU (*Remote Transport Unit* – Unidade de transporte Remota) ou Modbus-ASCII (*American Standard Class II*) é utilizado quando o meio físico é o RS-232 ou o RS-485. A Figura 2.16 apresenta a estrutura do frame Modbus RTU e ASCII (MODBUS, 2014).

ADU (Aplication data unit ou Unidade de dados da aplicação) ou Frame Modbus				
	Endereço do equipamento	Código da função	Dados	Checagem de erro
RTU	1 byte	1 Byte	0 a 252 bytes	2 bytes CRC
ASCII	2 Caracteres	2 caracteres	0 a 252 caracteres	2 caracteres LRC
PDU (Protocol data unit)				

Figura 2.16 – Frames Modbus-RTU / ASCII

Fonte: AUTOR, 2015.

Quando utilizado o meio físico *Ethernet*, são adicionados os cabeçalhos do protocolo TCP no frame de dados Modbus. A Figura 2.17 apresenta o frame do Modbus-TCP.

Frame Modbus TCP					
MBAP				Código da função	Dados
Transaction Identifier	Protocol Identifier	Length	Unit ID		
2 bytes	2 bytes	2 bytes	1 byte	1 Byte	0 a 252 bytes

Figura 2.17– Frame Modbus-TCP

Fonte: AUTOR, 2015.

As requisições do mestre aos escravos, possuem numeração padronizada e especificada no protocolo Modbus (MODBUS, 2014). As funções principais em hexadecimal são:

Leitura:

- 0x01-*Read Coils* (Ler saídas digitais);
- 0x02-*Read Discrete Inputs* (Ler entradas digitais);
- 0x03-*Read Holding Registers* (Ler saídas analógicas ou registradores internos);
- 0x04-*Read Input Registers* (Ler entradas analógicas).

Escrita Unitária:

- 0x05-*Write Single Coil* (Escrever em uma saída digital);
- 0x06-*Write Single Register* (Escrever em uma saída analógica ou registrador interno).

Escrita Múltipla:

- 0x15-*Write Multiple Coils* (Escrever em múltiplas saídas digitais);
- 0x16-*Write Multiple Registers* (Escrever em múltiplas saídas analógicas ou registradores internos).

3 PROJETO E IMPLEMENTAÇÃO

3.1 ABORDAGEM DO SISTEMA PROPOSTO

O sistema proposto é composto de 3 partes: módulo eletrônico, comunicação e *software*. Primeiro, o módulo eletrônico tem a função de efetuar a leitura de dois sensores de vazão e nível e armazenar estes valores em uma unidade de memória. O módulo eletrônico possui recurso para configuração de relógio com data e hora dia da semana e permite conexão com *display* e botões para permitir a um usuário navegar por um *menu* de visualização e programação de parâmetros.

A comunicação proposta do módulo eletrônico com o computador tem o objetivo de monitorar valores tratados dos sensores no campo, receber estes valores e realizar um armazenamento em base de dados no computador. A diversidade de protocolos de comunicação do sistema proposto, permite visualização em computador fixo e meio portátil. Assim, possibilita o *software* proposto gerar gráficos e criar tabelas em base de dados. A Figura 3.1 ilustra o diagrama do sistema proposto.

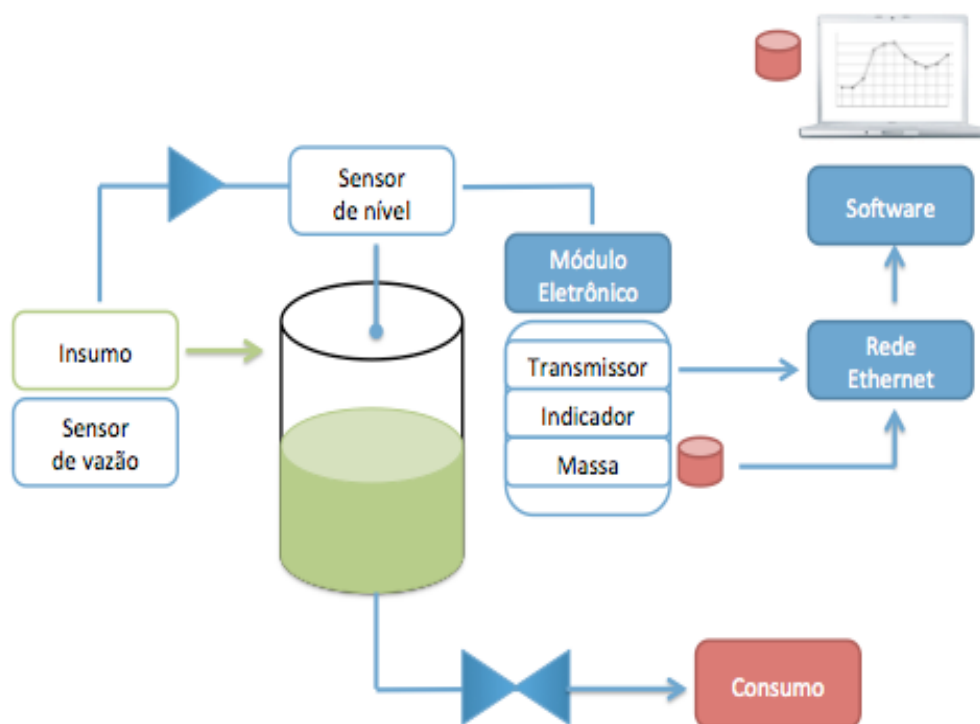


Figura 3.1 – Diagrama de funcionamento do sistema proposto.

Fonte: AUTOR, 2014.

2.2 HARDWARE

O Módulo eletrônico e seu *Firmware* (*Software* embarcado) associado, foram projetados para seguir a função descrita na Figura 3.2.

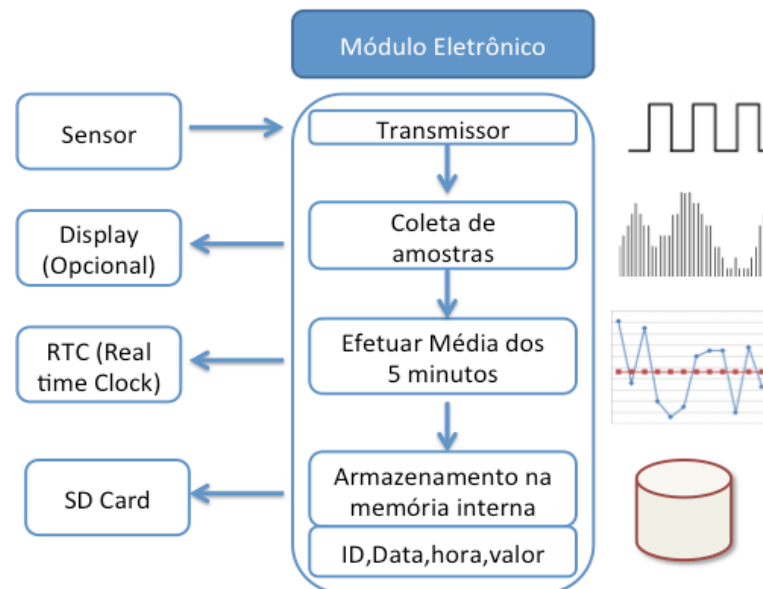


Figura 3.2 – Diagrama de funcionamento do hardware.

Fonte: AUTOR, 2014.

É composto de um *Arduino Mega* com demais elementos sobrepostos (*Shields*) conforme mostra a Figura 3.3.

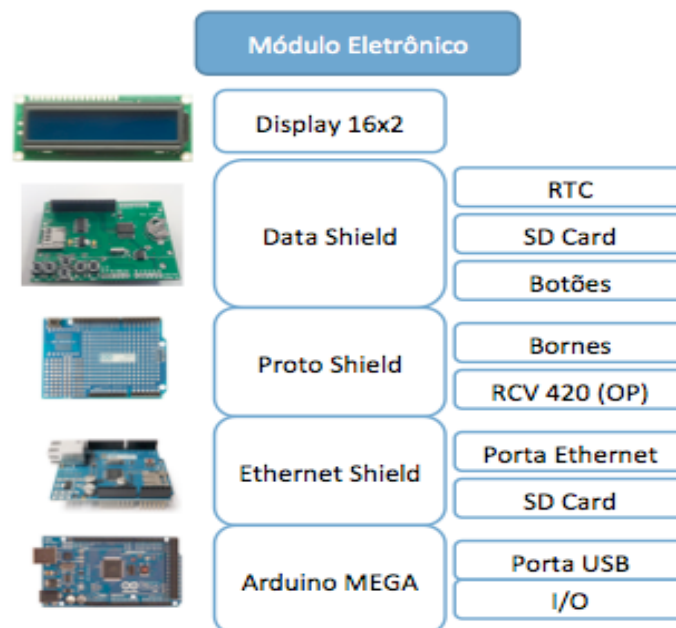


Figura 3.3 – Sobreposição de montagem do hardware.

Fonte: AUTOR, 2014.

O *Display* utilizado e suas funcionalidades conforme Figura 3.4 ilustra, se configura de forma matricial 16x2 Lumex, com pinagem posicionada para encaixe no *shield* de dados, conforme apresentado na Figura 3.3

Display 16 x 2	
Pino	Função
1	Vss (Gnd)
2	Vdd (5V)
3	Contraste (resistor 10k no Gnd)
4	Seleção de registrador
5	Leitura/Escrita (Gnd)
6	Habilitar
7	DO (N.U.)
8	D1 (N.U.)
9	D2 (N.U.)
10	D3 (N.U.)
11	D4
12	D5
13	D6
14	D7
15	A (5V)
16	K (Gnd)

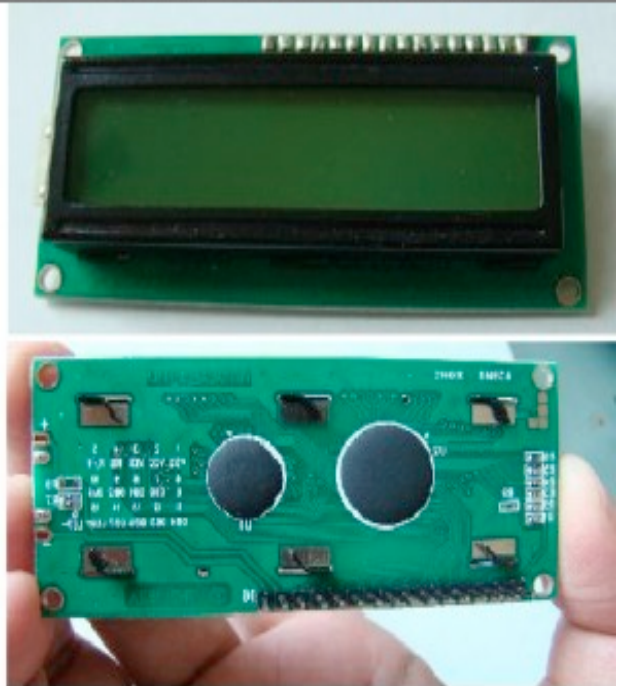


Figura 3.4 – Display serial 16x2.

Fonte: AUTOR, 2014.

Foi utilizado para o projeto e validação do hardware e software, sensor HC-RS04 e sensor YF-S201, cujas funcionalidades a Figura 3.5 apresenta.

	
HC-RS04 - Sensor de Ultrassom	
Pino	Função
1	Vss (5V)
2	Trigger
3	Echo
4	Vdd (Gnd)

	
YF-S201 - Sensor de Vazão	
Pino	Função
1	Vss (5V)
2	Sinal
3	Vdd (Gnd)

Figura 3.5 – HC-RS04 e YF-S201: Sensores de nível e vazão.

Fonte: AUTOR, 2014.

O *Arduino Mega* Revisão 3, baseado no microcontrolador ATMEGA 2560, foi utilizado como escudo principal do módulo e escolhido por, além de apresentar os recursos necessários à implementação do sistema proposto, a capacidade da memória de programa interna foi suficiente para abarcar o código do firmware necessário à aplicação, afim de executar as funções predeterminadas em código. Suas funcionalidades são apresentadas na Figura 3.6

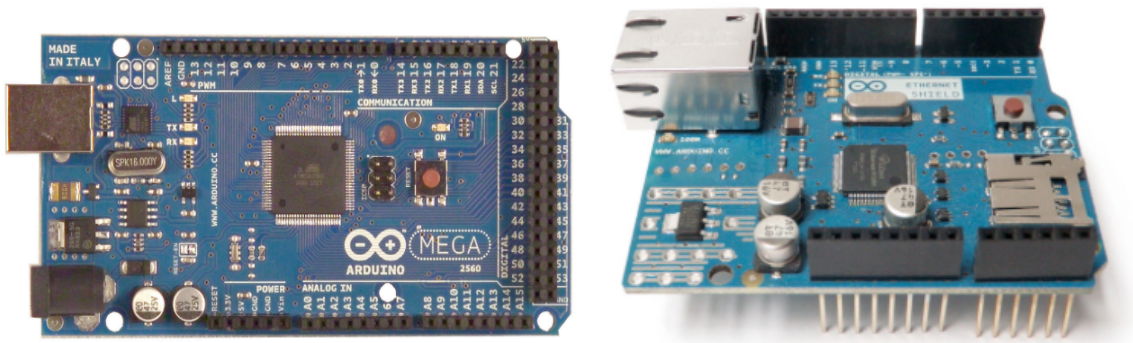


Figura 3.6 – *Arduino e Ethernet shield*.

Fonte: AUTOR, 2014.

O *Shield* de dados utilizado, ilustrado na Figura 3.7, possui um relógio de tempo real alimentado por bateria, possui barramento para cartão de memória, além de botões de navegação e soquete para conexão simples do display apresentado na Figura 3.4.

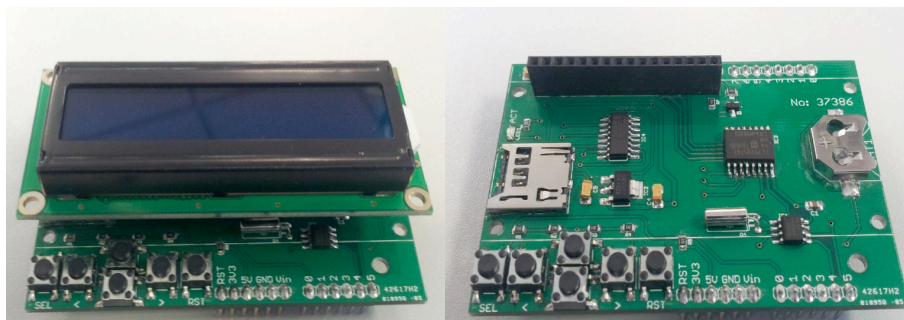


Figura 3.7 – *Data Shield*.

Adaptado de: LABDEGARAGEM, 2014.

2.3 COMUNICAÇÃO

Para a comunicação entre o hardware e o software, foi utilizado o protocolo Ethernet, através dos serviços disponibilizados como UDP e TCP, sendo este último implementado o protocolo aberto Modbus-TCP. A Figura 3.8 ilustra.

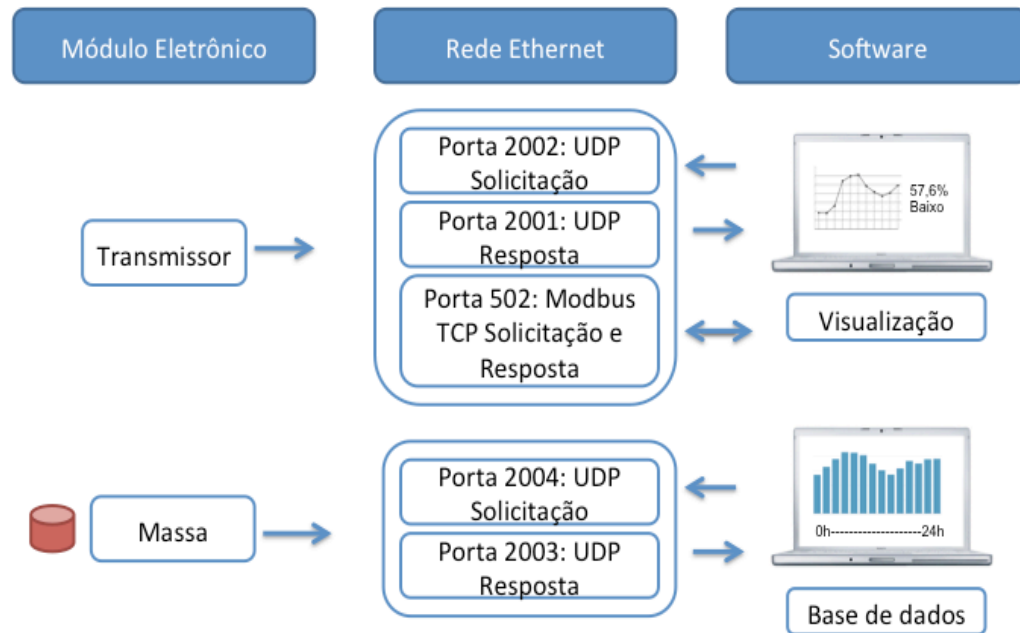


Figura 3.8 – Diagrama do projeto de comunicação do sistema proposto.

Fonte: AUTOR, 2014.

Para viabilidade da comunicação UDP, foi desenvolvido um protocolo proprietário específico onde existe uma tabela de funções cuja requisição enviada pela porta 2002, responde pela porta 2001.

Ademais, o protocolo Modbus-TCP tem como padrão a utilização da porta 502 tanto para envio quanto para recebimento de dados, operando também em um modo de requisição e resposta.

No caso específico da solicitação de leitura de memória de massa, foram utilizadas as portas 2004 e 2003 respectivamente para requisição e resposta, também operando no modo de requisição e resposta.

2.4 SOFTWARE

Para desenvolvimento do software, foi utilizado o Visual Basic 2008 e seus respectivos acessórios de programação. A Figura 3.9 apresenta o projeto proposto.

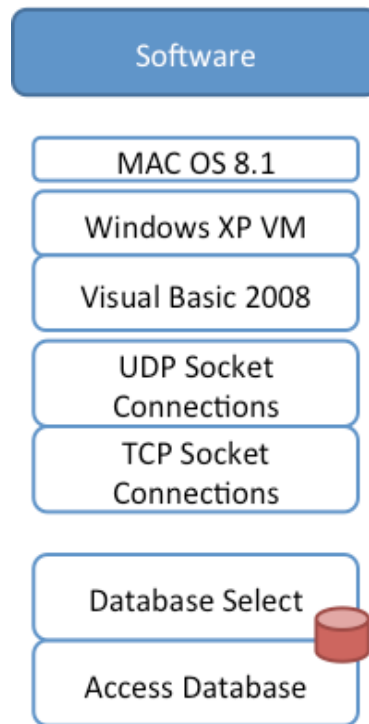


Figura 3.9 – Diagrama do projeto do software proposto.

Fonte: AUTOR,2014.

Através de um *Macbook* com sistema operacional MAC OS 8.1, e com uma máquina virtual do sistema operacional Microsoft Windows XP, foi desenvolvido o aplicativo em linguagem *Basic*.

Para a comunicação com o *Arduino* foram utilizadas as estâncias Socket connections para UDP e TCP, disponíveis para a corrente versão do Visual Basic.

A base de dados utilizada para armazenar os dados obtidos da aquisição do sinal analógico do sensor conectado ao *Arduino*, foi a *Microsoft Access*, por ser de mais simples implementação.

4 AVALIAÇÃO DE RESULTADOS EXPERIMENTAIS

4.1 MÉTODO DA EXPERIMENTAÇÃO

Para validar o modelo proposto no Capítulo 3, ilustrado pela Figura 3.1, foi utilizada a seguinte sequência de experimentos:

- Construção de protótipo para validação de conceito;
- Verificação da variação do valor de nível;
- Verificação da variação do valor de vazão;
- Teste comunicação UDP;
- Teste de comunicação Modbus-TCP;
- Teste de armazenamento de dados;

4.2 INFRA-ESTRUTURA DE TESTES

Foi desenvolvido um protótipo para a validação do modelo descrito na Figura 3.1. para realizar todos os testes descritos em 4.1. A Figura 4.1 apresenta o protótipo montado em pequena escala do sistema proposto. Ele contém um roteador WiFi, bomba para circulação de água, dois reservatórios (superior e inferior) módulo eletrônico e sensores de nível HC-SR04 e vazão YF-S201.

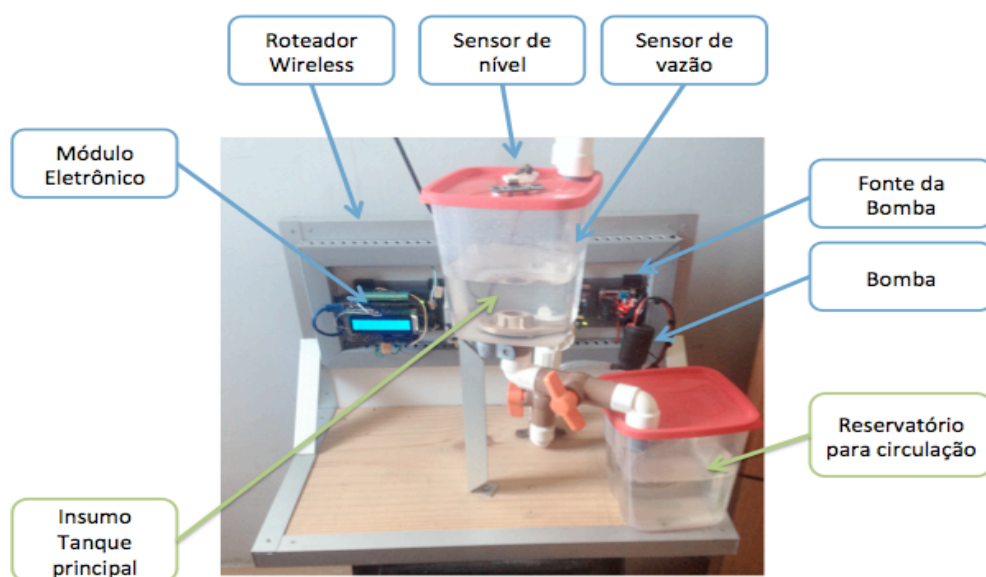


Figura 4.1 – Protótipo do sistema proposto.

Fonte: AUTOR,2014.

Primeiramente foi utilizado o sensor HC-SR04 para medir a variação de nível no tanque experimental graduado. O código para leitura foi desenvolvido com base no diagrama de blocos da estrutura de teste é apresentado pela Figura 4.2

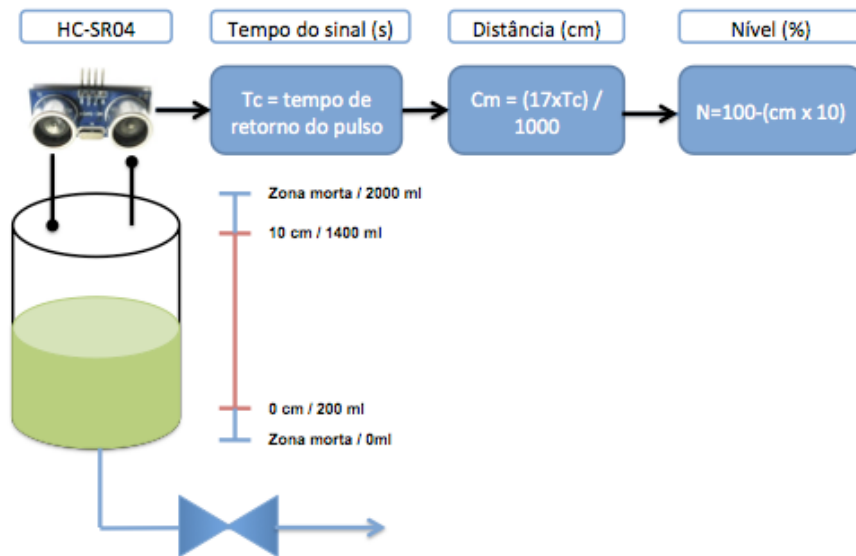


Figura 4.2 – Validação da leitura de sensor de nível.

Fonte: AUTOR,2015.

Posteriormente foi adotado princípio similar para a validação do sensor YF-S201, de forma que ele possa aferir a vazão em Litros por hora de entrada de água no tanque experimental graduado, conforme mostra a Figura 4.3.

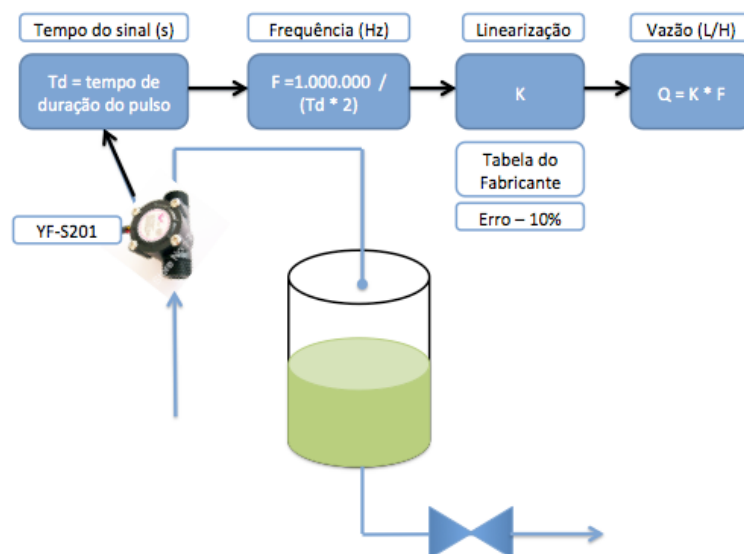


Figura 4.3 – Validação da leitura de sensor de vazão tipo turbina.

Fonte: AUTOR,2015.

Por fim, foi implementado um roteador WiFi para comunicação do *Arduino* com Notebook e Smartphone a fim de validar a comunicação UDP e Modbus-TCP, conforme ilustrado na Figura 4.4.

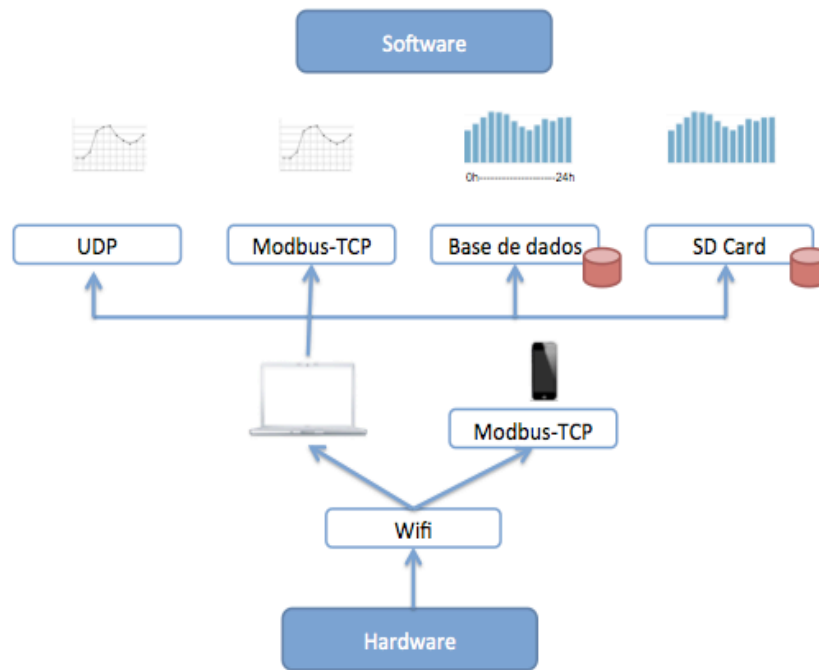


Figura 4.4 – Validação da comunicação UDP e Modbus-TCP.

Fonte: AUTOR,2015.

O modelo proposto pode ser adequado a qualquer uma das técnicas de medição descritas no Capítulo 2. Os sensores escolhidos para validar o modelo proposto no capítulo 3, foram o sensor de nível por ultrassom e o sensor de vazão do tipo turbina pelo fato dos mesmos serem de fácil aquisição e possibilitarem integração simplificada com o protótipo do módulo eletrônico apresentado na Figura 4.1

A infraestrutura de testes apresentada na Figura 4.4 com o intuito de validar a comunicação UDP, WiFi e Modbus-TCP foi proposta para utilização em UDP para o computador por causa da maior velocidade com que o protocolo UDP permite na troca de dados em relação ao TCP. O Modbus-TCP foi testado em um equipamento portátil em virtude da maior facilidade em obter aplicativos de uso gratuito na *internet* que possibilitem testes com este protocolo.

4.3 RESULTADOS DOS TESTES

Primeiro, ao posicionar o sensor de nível é possível obter a distância em centímetros do sensor à superfície do fluido. Posteriormente, é possível converter este valor de distância em percentual, tendo como base as distâncias máximas e mínimas da zona morta do fundo do tanque até a condição de maior nível possível de água que o sistema pode fornecer. Assim é possível tomar como base a escala graduada do tanque e verificar o percentual de nível de fluido no tanque experimental.

Os resultados dos testes para o sensor de nível estão apresentados na Tabela 4.1. A variação do erro de medição comparado ao valor obtido através de uma régua, segue ilustrado pela Figura 4.5.

Tabela 4.1 – Variação de nível

Fonte: AUTOR, 2015.

Graduação (ml)	Distância (cm)	Nível (%)	Medido (cm)	Erro
0	-	-	-	-
100	-	-	-	-
200	-	-	-	-
300	10	0,00	10	0
400	9	7,14	9,2	-0,2
500	8	14,29	8,3	-0,3
600	7	21,43	7,2	-0,2
700	7,5	28,57	7,6	-0,1
800	6	35,71	6,1	-0,1
900	6,5	42,86	6,6	-0,1
1000	5	50,00	5,2	-0,2
1100	4,5	57,14	4,6	-0,1
1200	4	64,29	4	0
1300	3	71,43	3,1	-0,1
1400	-	78,57	-	
1500	-	85,71	-	
1600	-	92,86	-	
1700	-	100,00	-	
1800	-	-	-	
1900	-	-	-	
2000	-	-	-	

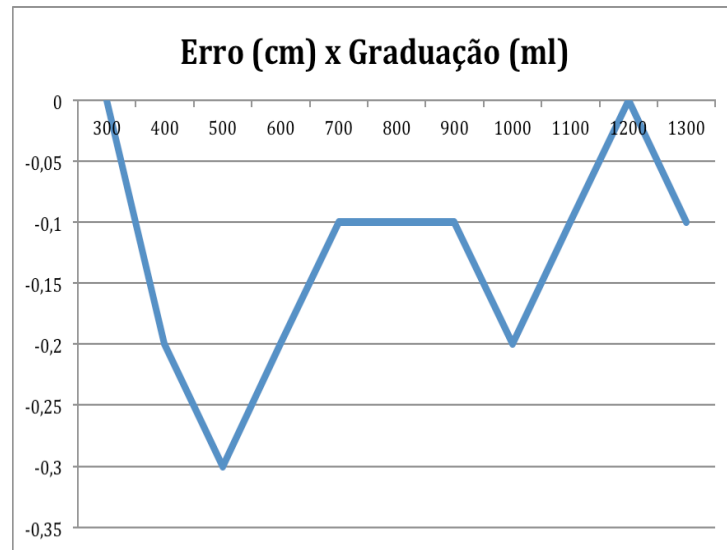


Figura 4.5 – Curva de erro versus graduação.

Fonte: AUTOR,2015.

O sensor de vazão YF-S201 é um sensor tipo turbina baseado no efeito Hall. Possui curva de vazão proporcional à frequência disponibilizada pelo fabricante que não passa pelo ponto zero. A Figura 4.6 apresenta o gráfico.

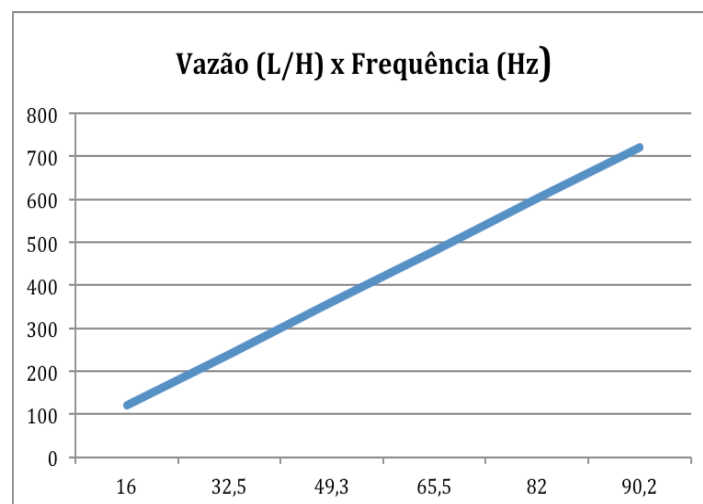


Figura 4.6 – Curva de vazão versus frequência.

Fonte: AUTOR,2015.

Para efetuar a medição do valor de vazão em L/H foi efetuada linearização da curva para obtenção de uma constante proporcional $K = 7,469$, baseada na média das diferenças de valores Q/F retirados da tabela disponibilizada pelo fabricante. A Tabela 4.2 apresenta estes valores.

Tabela 4.2 – Dados de vazão e frequência

Fonte: AUTOR,2015.

Frequência (Hz)	Vazão (L/h)	Vazão = frequência . X	Erro
16	120	7,500000000	0,030929049
32,5	240	7,384615385	-0,084455567
49,3	360	7,302231237	-0,166839714
65,5	480	7,328244275	-0,140826677
82	600	7,317073171	-0,151997781
90,2	720	7,982261641	0,513190689
Erro		X Média	
10%		7,469070951	

Conforme os dados obtidos do experimento e o valor determinado para K. foi obtido o valor de vazão nominal da bomba de teste em aproximadamente 8 Hz, 56 L/H.

O sensor de vazão apresenta curva de desvio do erro médio com crescimento em função do aumento da frequência. Esta característica indica que o sensor opera melhor na medição de valores menores de vazão. A Figura 4.7 ilustra esta característica do equipamento.

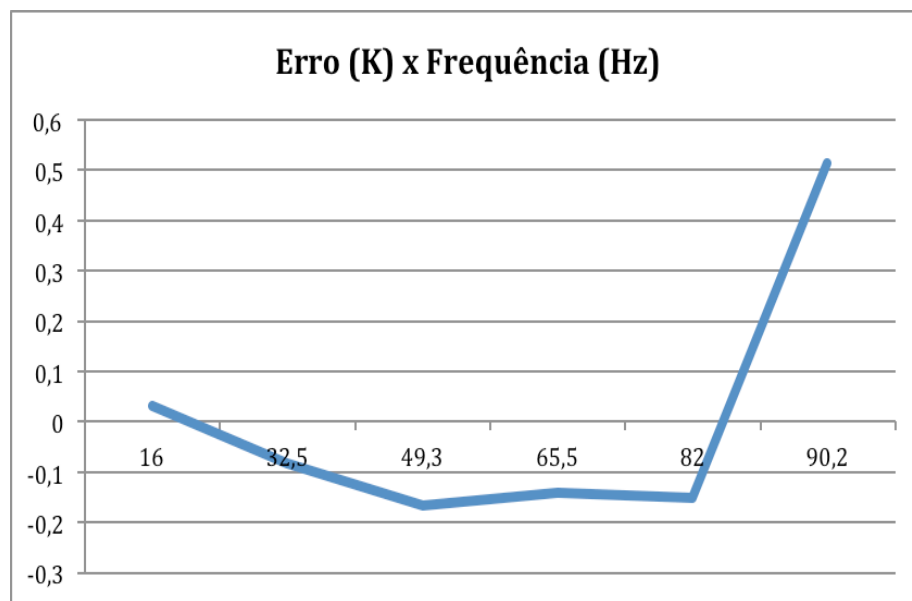


Figura 4.7 – Curva de desvio do erro médio em função da frequência.

Fonte: AUTOR,2015.

Para validar o protocolo de comunicação para UDP pela porta 2001 e 2002, foi utilizado o próprio hardware conectado ao software, foi programado o endereço IP 192.168.0.10 e foi testado o envio de pacotes UDP. A Figura 4.8 ilustra o resultado.

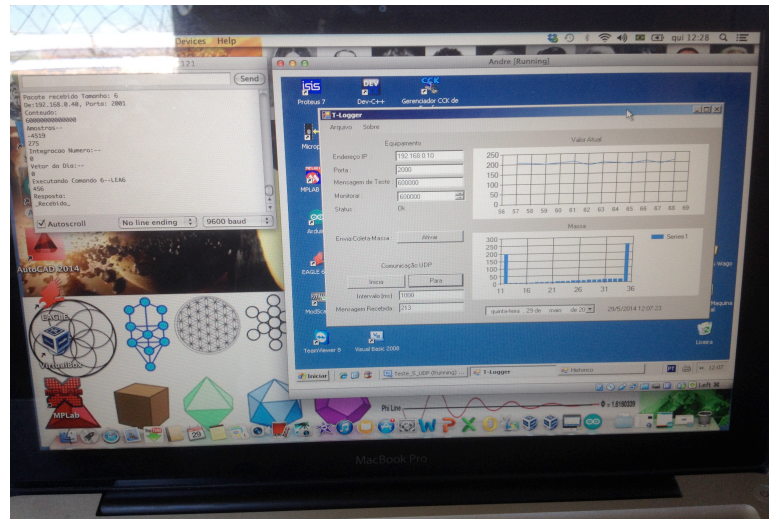


Figura 4.8 – Validação da comunicação UDP

Fonte: AUTOR,2015.

Para validar o protocolo Modbus-TCP implementado, foi utilizado o software Mod-view em um Iphone, em comunicação direta com o hardware proposto. A Figura 4.9 apresenta o resultado obtido deste teste.

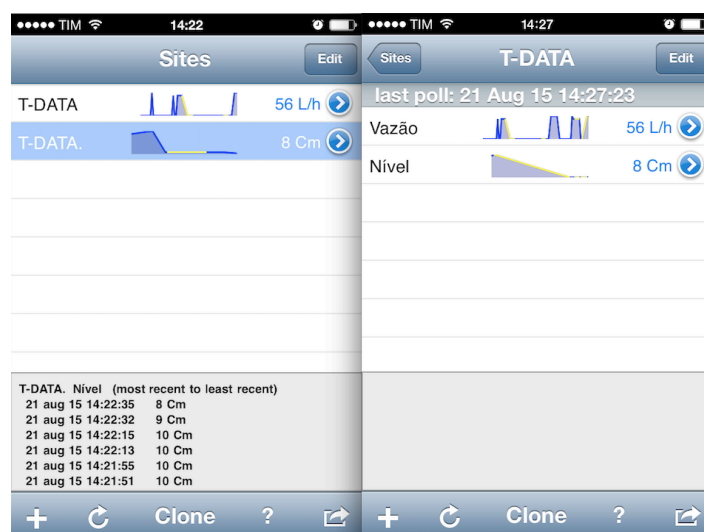


Figura 4.9 – Validação da comunicação Modbus-TCP.

Fonte: AUTOR,2014.

Para teste da aquisição de dados, foi iniciada a comunicação com o hardware e foi habilitada a aba que possui a função de aquisição de dados a enviar para uma base de dados access, conforme apresentado na Figura 4.10.

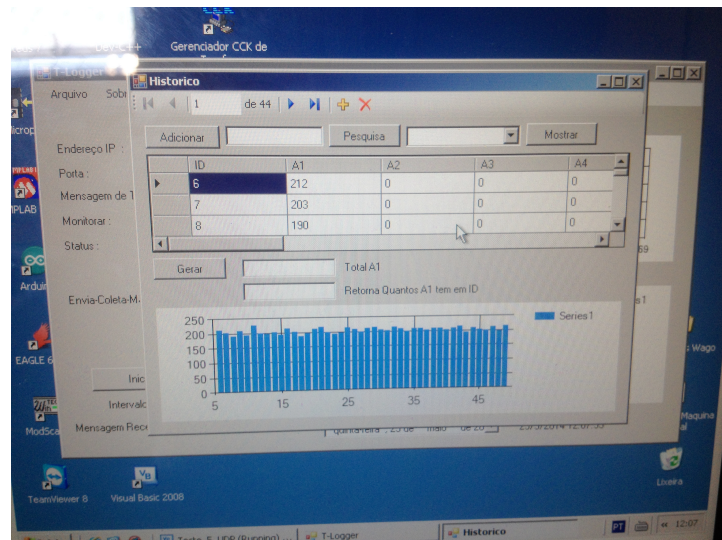


Figura 4.10 – Validação da leitura da base de dados.

Fonte: AUTOR,2014.

Para validação da memória de massa, foi iniciado o sistema e o mesmo foi programado para efetuar armazenamento de data, hora, dia da semana, valor de vazão e valor de nível a cada 10 segundos no cartão SD. A Figura 4.11 apresenta o cartão utilizado e o resultado deste teste.

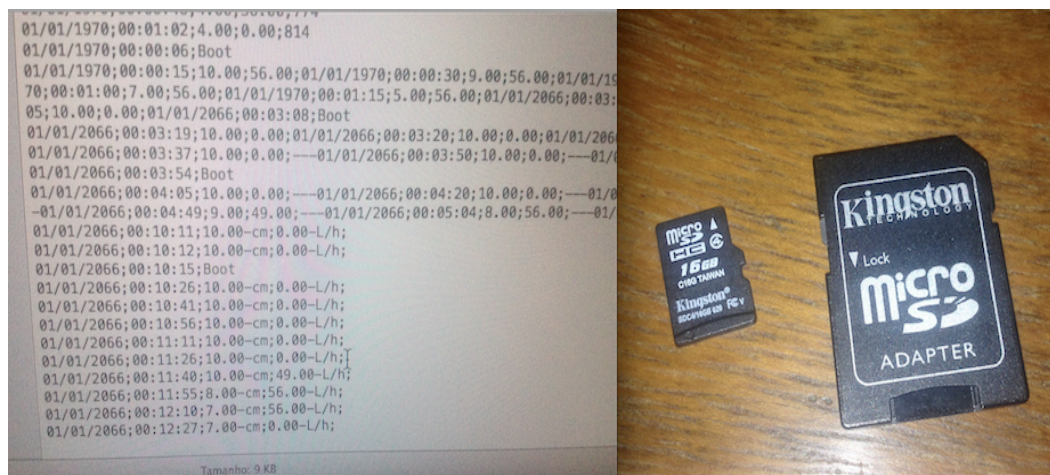


Figura 4.11 – Validação da leitura da memória de massa.

Fonte: AUTOR,2015.

4.4 AVALIAÇÃO DOS RESULTADOS

Os resultados obtidos dos testes realizados com o sensor de nível dimensionado para o tanque experimental graduado foram satisfatórios com erro global em torno de 5 % na faixa de operação entre 200 ml e 1400 ml.

Os resultados obtidos para a linearização do sensor de vazão dimensionado para o tanque de nível experimental, apresentou resultado satisfatório com erro global em torno de 10 % de tabela e o erro relativo a desvio da média crescente em função do aumento da frequência, de 0,03 a 0,513. A bomba de teste apresentou vazão nominal de 56 L/H, 8 Hz.

A comunicação UDP ocorreu como esperado no endereço IP 192.168.0.10, porta 2001 e 2002, para envio do valor de nível e vazão através da rede *Ethernet*. A comunicação Modbus-TCP foi realizada com sucesso através da implementação de biblioteca Modbus para o Arduino Mega utilizando a porta especificada 502 para envio e recebimento de dados através de um *Smartphone*. O teste de armazenamento de dados de data, hora, dia da semana, vazão e nível no cartão de memória obteve resultado satisfatório conforme apresentado na Figura 4.11.

A diferença simples de valores entre o projeto proposto e a aquisição de dispositivos prontos no mercado que visam fazer função similar, apresenta variação de custo média de quatro vezes se comparada com o sistema proposto, como apresenta a Tabela 4.3.

Tabela 4.3 – Comparativo de custos entre equipamentos

Fonte: AUTOR,2015.

Equipamento	Fabricante	Valor
5800	CCK	R\$2.850,00
<i>Field Logger</i>	<i>Novus</i>	R\$2.480,00
Sistema Proposto	Autor do Presente Trabalho	R\$ 630,00

Entretanto, estes comparativos não levam em consideração custos com invólucro, categoria de proteção, mão de obra produtiva, certificações técnicas dos produtos, transporte e impostos.

5 CONCLUSÃO

O Trabalho realizado, documentado e aqui apresentado revelou tanto uma natureza multidisciplinar quanto interdisciplinar. A natureza multidisciplinar envolve a consideração de conhecimentos relacionados à diversas áreas de conhecimento como, por exemplo, no caso deste Trabalho: automação industrial, protocolos de comunicação, redes industriais, desenvolvimento de *software*, gestão de insumos industriais, plataformas eletrônicas *Open Source*, processamento e armazenamento de dados, Física dos Fluidos, elétrica e eletrônica, dentre outras competências. A interdisciplinaridade refere-se à consideração desses conhecimentos multidisciplinares aplicados de forma integrada. Desta, o desenvolvimento de soluções para Automação Industrial deve ocorrer segundo os preceitos de Engenharia Simultânea (BAXTER, 1998).

Assim, com este Trabalho, foi apresentado um modelo e desenvolvido um protótipo de testes de um sistema de medição de nível e vazão de fluido com o intuito de validar de forma experimental e em menor escala o conceito proposto no Trabalho. Conforme demonstrado e explicado anteriormente, para a montagem deste protótipo de testes foi necessário utilizar diversos elementos e conhecimentos específicos de eletrônica, elétrica, automação, linguagens de programação e ciência da computação, dada a característica interdisciplinar deste modelo de Trabalho.

Os resultados obtidos experimentalmente revelaram resultados que corroboraram com as expectativas inicialmente previstas para o comportamento funcional do conceito proposto. Um deles diz respeito ao sensor de nível. Este operou de forma correta e apresentou resultados satisfatórios, para um ensaio em um tanque com graduação 0 a 2000 ml e faixa de operação entre 200 e 1400 ml. Outro concerne aos resultados acerca dos sensores de nível e de vazão. Durante o desenvolvimento, testes e calibração desses sensores, foram determinadas curvas de teste que verificaram erros relativos de medição o qual se verificou que operavam de forma satisfatória para pequenas distâncias e pequenas vazões. As características do sensor de nível mostram tendência a uma faixa de erro de medição constante em função da variação de nível enquanto que as do sensor de vazão mostram que seu desvio do valor médio aumenta proporcionalmente ao aumento da frequência.

A comunicação entre o sistema proposto e o computador e o smartphone ocorreram de forma satisfatória tanto para o caso do protocolo UDP quando para o caso do protocolo Modbus-TCP. Pacotes foram enviados e recebidos no endereço de rede especificado pelo projeto e pelos respectivos protocolos de comunicação. O armazenamento de valores de vazão e nível, data, hora e dia da semana no cartão de memória SD, foi realizado de forma satisfatória.

Durante o desenvolvimento do projeto, foram implementados diversos algoritmos em linguagem C e Basic, sendo necessário estudo e aprendizado de funções de microcontroladores e computadores, assim, os objetivos específicos foram alcançados.

Das muitas possibilidades de temas como desdobramentos futuros para este Trabalho, tem-se:

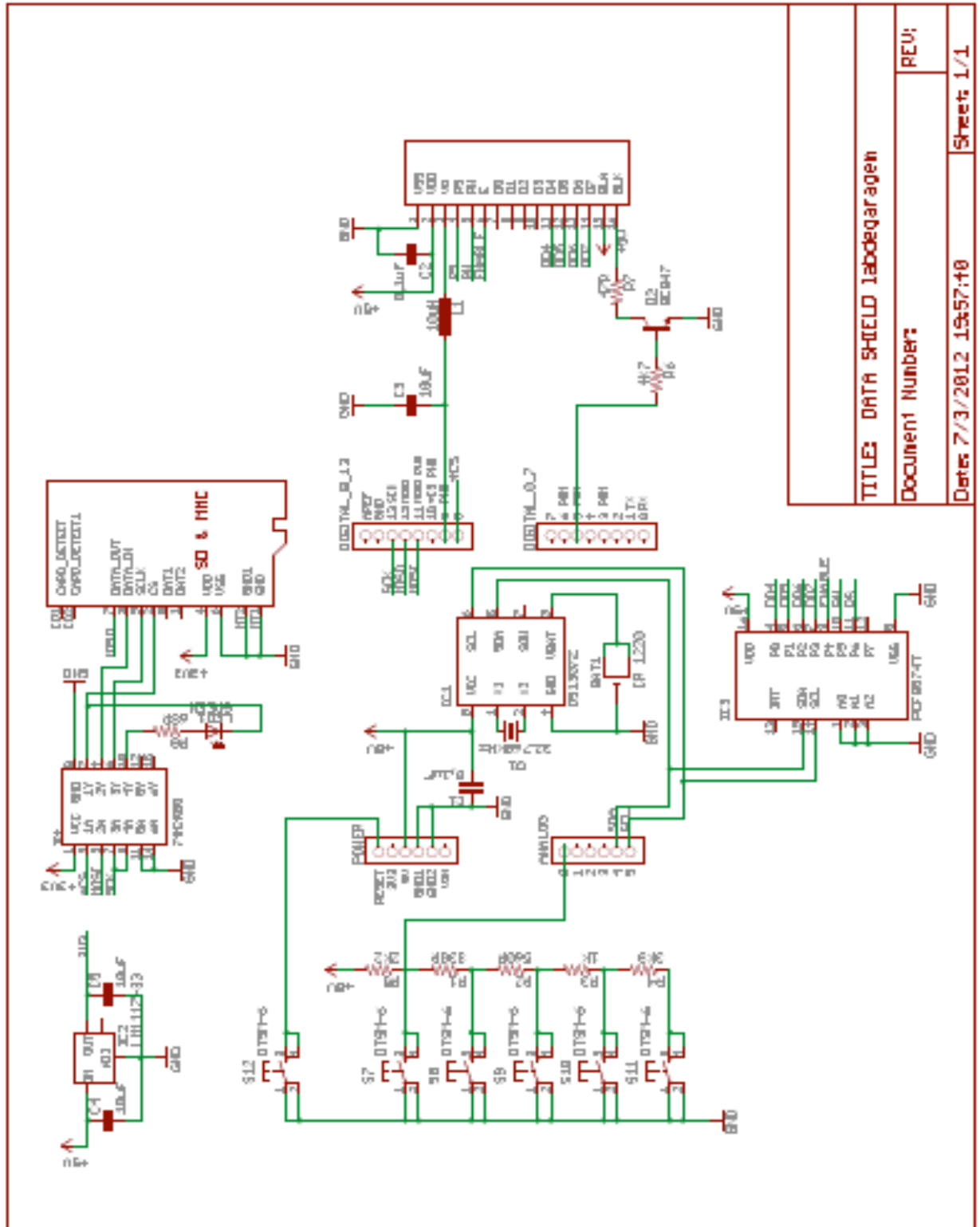
- Desenvolvimento de *protoshield* expansivo para o *Arduino Mega*;
- Desenvolvimento de invólucro de PVC com suporte para trilho DIN;
- Desenvolvimento de *protoshield* com circuito integrado RCV 420 para medição de sensores 4-20 mA;
- Desenvolvimento de tela de supervisão em software supervisor para teste e simulação de alarmes;
- Ampliação da função do menu principal para possibilitar configuração de endereço IP e parâmetros de sensores pelo usuário ao invés de configuração pelo código;
- Substituição da bomba por uma de maior potência e de corrente contínua, possibilitando assim variar a vazão pelo controle da tensão de armadura do estator, para melhor testar o comportamento do modelo do sistema proposto.

A validação de um modelo em menor escala de um sistema eletrônico, que tenha o objetivo de medir vazão e nível pode, a partir destas propostas de trabalhos e testes futuros, criar bases para estudo e desenvolvimento de uma alternativa futura com tecnologias abertas para medição em instalações que já possuam um instrumento de campo adequado para o campo e que também tenham condições para dispor a eletrônica em um ambiente de testes controlado.

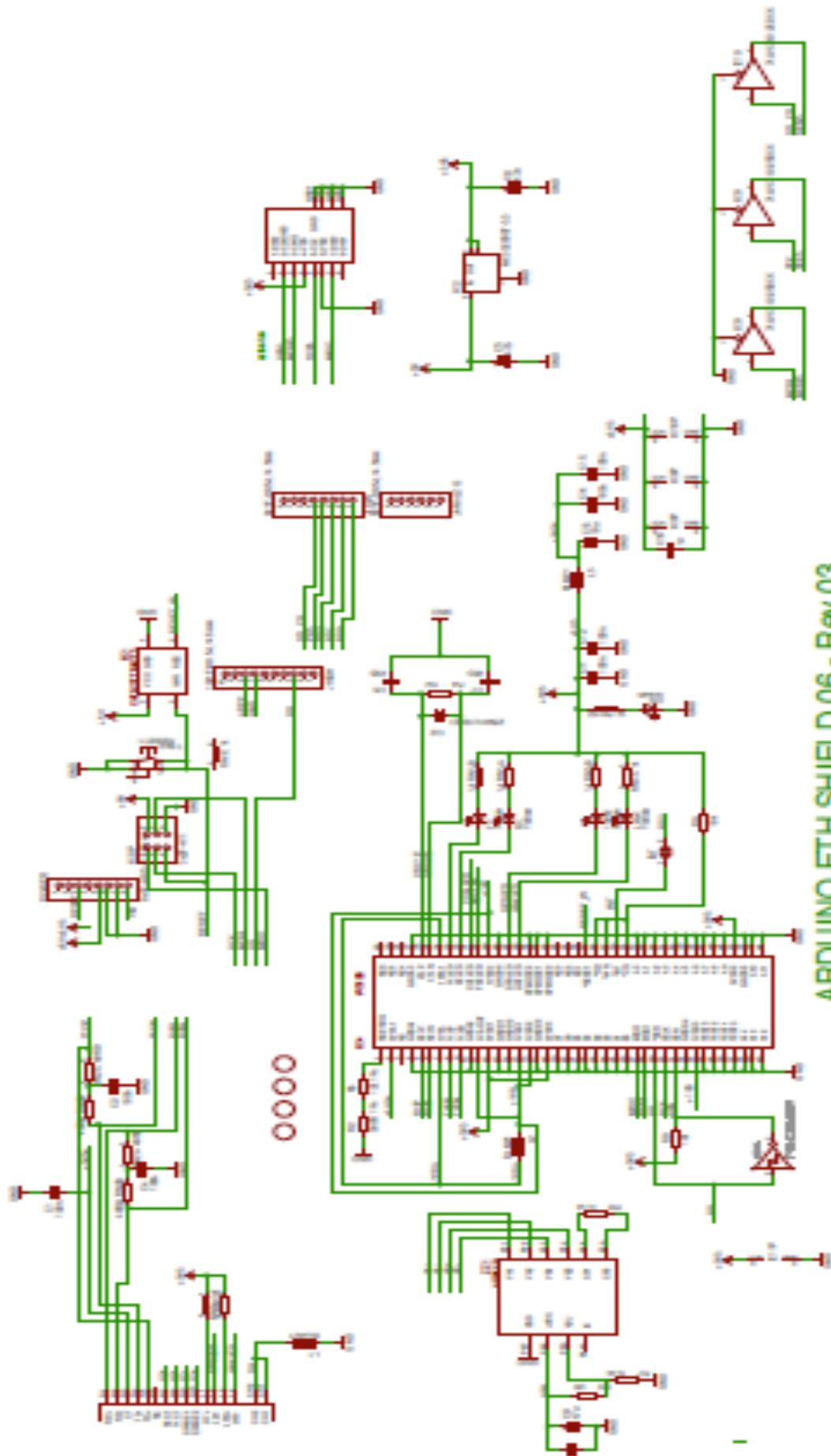
ANEXOS

Anexo A – Esquemático do *Data Shield*.

Fonte: (LABORATORIO DE GARAGEM, 2015).



Anexo B – Esquemático do *Ethernet Shield* Rev 03. Fonte: (ARDUINO, 2014).



Intellect's Designs ARE PROVIDED "AS IS" AND, WITH ALL FAULTS, AND NO OTHER WARRANTIES, EXPRESS OR IMPLIED, REGARDING PRODUCTS, INCLUDING BUT NOT LIMITED TO, ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Intellect may make changes to specifications and product configurations at any time without notice. The Customer must not rely on the absence or disavowal of any features or functions made of "new" or "old" Intellect's. Intellect reserves the right to change specifications and shall have no responsibility for any changes or omissions from time to time. The product information on the Web Site or Materials is subject to change without notice. Do not base a design on this information. ARDUINO is a registered trademark.

Anexo C – Datasheet do ATMEGA 2560. Fonte: (ATMEGA 2560, 2015)



Atmel ATmega640/V-1280/V-1281/V-2560/V-2561/V

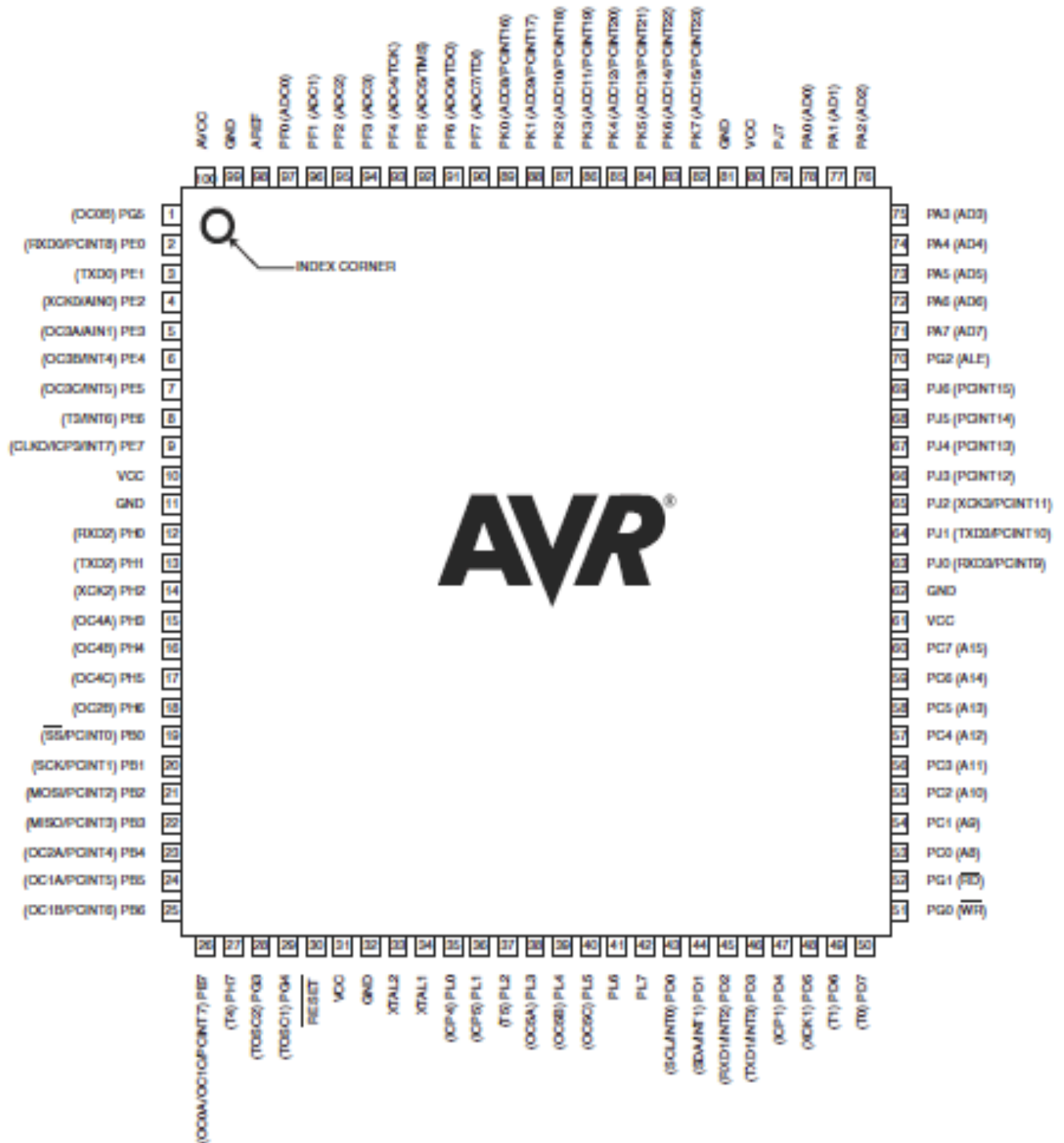
8-bit Atmel Microcontroller with 16/32/64KB In-System Programmable Flash

DATASHEET

Features

- High Performance, Low Power Atmel® AVR® 8-Bit Microcontroller
- Advanced RISC Architecture
 - 135 Powerful Instructions – Most Single Clock Cycle Execution
 - 32 x 8 General Purpose Working Registers
 - Fully Static Operation
 - Up to 16 MIPS Throughput at 16MHz
 - On-Chip 2-cycle Multiplier
- High Endurance Non-volatile Memory Segments
 - 64K/128K/256KBytes of In-System Self-Programmable Flash
 - 4Kbytes EEPROM
 - 8Kbytes Internal SRAM
 - Write/Erase Cycles: 10,000 Flash/100,000 EEPROM
 - Data Retention: 20 years at 85°C/ 100 years at 25°C
 - Optional Boot Code Section with Independent Lock Bits
 - In-System Programming by On-chip Boot Program
 - True Read-While-Write Operation
 - Programming Lock for Software Security
 - Endurance: Up to 64Kbytes Optional External Memory Space
- Atmel® QTouch® library support
 - Capacitive touch buttons, sliders and wheels
 - QTouch and QMatrix acquisition
 - Up to 64 sense channels
- JTAG (IEEE® std. 1149.1 compliant) Interface
 - Boundary-scan Capabilities According to the JTAG Standard
 - Extensive On-chip Debug Support
 - Programming of Flash, EEPROM, Fuses, and Lock Bits through the JTAG Interface
- Peripheral Features
 - Two 8-bit Timer/Counters with Separate Prescaler and Compare Mode
 - Four 16-bit Timer/Counter with Separate Prescaler, Compare- and Capture Mode
 - Real Time Counter with Separate Oscillator
 - Four 8-bit PWM Channels
 - Six/Twelve PWM Channels with Programmable Resolution from 2 to 16 Bits (ATmega1281/2561, ATmega640/1280/2560)
 - Output Compare Modulator
 - SPI-channel, 10-bit ADC (ATmega1281/2561, ATmega640/1280/2560)
 - Two/Four Programmable Serial USART (ATmega1281/2561, ATmega640/1280/2560)
 - Master/Slave SPI Serial Interface
 - Byte Oriented 2-wire Serial Interface
 - Programmable Watchdog Timer with Separate On-chip Oscillator
 - On-chip Analog Comparator
 - Interrupt and Wake-up on Pin Change
- Special Microcontroller Features
 - Power-on Reset and Programmable Brown-out Detection
 - Internal Calibrated Oscillator
 - External and Internal Interrupt Sources
 - Six Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, Standby, and Extended Standby
- I/O and Packages
 - 64/65 Programmable I/O Lines (ATmega1281/2561, ATmega640/1280/2560)
 - 64-pad QFNMLF, 64-lead TQFP (ATmega1281/2561)
 - 100-lead TQFP, 100-ball BGA (ATmega640/1280/2560)
 - RoHS/Fully Green
- Temperature Range:
 - -40°C to 85°C Industrial
- Ultra-Low Power Consumption
 - Active Mode: 1MHz, 1.8V: 500µA
 - Power-down Mode: 0.1µA at 1.8V
- Speed Grads:
 - ATmega640/ATmega1280/ATmega1281V:
 - 0 - 4MHz @ 1.8V - 5.5V, 0 - 8MHz @ 2.7V - 5.5V
 - ATmega2560/ATmega2561V:
 - 0 - 2MHz @ 1.8V - 5.5V, 0 - 8MHz @ 2.7V - 5.5V
 - ATmega640/ATmega1280/ATmega1281:
 - 0 - 8MHz @ 2.7V - 5.5V, 0 - 16MHz @ 4.5V - 5.5V
 - ATmega2560/ATmega2561:
 - 0 - 16MHz @ 4.5V - 5.5V

Figure 1-1. TQFP-pinout ATmega640/1280/2560



APÊNDICE

Apêndice A: Código fonte do Microcontrolador Arduino ATMEGA 2560, Arduino IDE 1.0.5.

```
//-----
// Andre Pinto Marotta - 25-08-2015.
// hardware T-Data Firmware Versao 1.0.0
// Salvador, Bahia, Brasil
// Projeto de Hardware Open Source para registro de vazao e nivel
// IDE: Arduino 1.0.5 SO: MAC OSX 9.8.8
//-----
//Creditos:
//Aluno: Andre Pinto Marotta
// Orientador: Prof. Me. Cleber Vinicius
// Coordenador: Prof. Oberdan Rocha
//-----
// ***** BIBLIOTECAS *****
//-----
//Bibliotecas - DATASHIELD
#include <LiquidCrystal_I2C.h>
#include <EEPROM.h>
#include <WProgram.h>
#include <avr/pgmspace.h>
#include <Wire.h>
#include <Time.h>
#include <TimeAlarms.h>
#include <DS1307RTC.h>    //Necessary to LCD
#include <SD.h>
//Bibliotecas - MODBUS-TCP e Ethernet-UDP
#include <SPI.h>
#include <Ethernet.h>
#include <EthernetUdp.h>  // biblioteca UDP: bjoern@cs.stanford.edu 12/30/2008
#include <Mudbus.h>
//-----
// ***** CONSTANTES *****
const int ERROR_WINDOW = 50;    // tolerance +/- this value for the button's analog
input
const int BUTTONDELAY = 50;     // button's debounce delay
const int BUTTONREPEAT = 200;   // repeat delay of the pressed buttons
const int TIMEOUT = 30000;      // timeout value to exit to default display in
milliseconds (remember that INT type max allowed value is 32767)
const int LEDPIN = 5;           // output pin for the PWM of the LCD backlight
```

```

const int CONTRASTPIN = 9;    // output pin for the PWM of the LCD contrast
const int SD_CS = 53; //OBSSSSS Pin 8 is the CS chip select of the SD memory card
on this Shield
const int BUTTONPIN = A0;    // input pin for the buttons
//-----
// ***** MODBUS-TCP *****
Mudbus Mb;
//-----
// ***** ETHERNET-UDP *****
char packetBuffer[UDP_TX_PACKET_MAX_SIZE]; // Buffer sistêmico que recebe o pacote
UDP
char ReplyBuffer[] = "_Recebido_";          // String de caracteres para enviar
pacote UDP
int  buffer[24];                            // String de Inteiros para enviar pacote
UDP
    unsigned int localPort = 2000;           // Porta local para troca de dados UDP
EthernetUDP Udp; //Instancia que permite troca de dados em UDP
//-----
// Memoria de massa: Contadores variaveis e String de armazenamento
//-----
int c=0;          // Contador do loop de integracao
int amostras=0;   // valor que armazena amostras coletadas - 1 amostra por segundo
int t=0;          // Conta o numero de integracoes por dia
int inicia;       // Inicia o armazenamento da memoria de massa
int para;         // Para o armazenamento da memoria de m
//-----
int dia[24];      // dia com 24 integracoes - 1 integracao por hora
int d1=dia[1]=200; // Valor da integracao 1/24 do dia
int d2=dia[2]=1;   // Valor da integracao 2/24 do dia
int d3=dia[3]=2;   // Valor da integracao 3/24 do dia
int d4=dia[4]=3;   // Valor da integracao 4/24 do dia
int d5=dia[5]=4;   // Valor da integracao 5/24 do dia
int d6=dia[6]=5;   // Valor da integracao 6/24 do dia
int d7=dia[7]=6;   // Valor da integracao 7/24 do dia
int d8=dia[8]=7;   // Valor da integracao 8/24 do dia
int d9=dia[9]=8;   // Valor da integracao 9/24 do dia
int d10=dia[10]=9; // Valor da integracao 10/24 do dia
int d11=dia[11]=10; // Valor da integracao 11/24 do dia
int d12=dia[12]=11; // Valor da integracao 12/24 do dia
int d13=dia[13]=12; // Valor da integracao 13/24 do dia
int d14=dia[14]=13; // Valor da integracao 14/24 do dia
int d15=dia[15]=14; // Valor da integracao 15/24 do dia
int d16=dia[16]=15; // Valor da integracao 16/24 do dia
int d17=dia[17]=16; // Valor da integracao 16/24 do dia
int d18=dia[18]=17; // Valor da integracao 18/24 do dia

```

```

int d19=dia[19]=18; // Valor da integracao 19/24 do dia
int d20=dia[20]=19; // Valor da integracao 20/24 do dia
int d21=dia[21]=20; // Valor da integracao 21/24 do dia
int d22=dia[22]=21; // Valor da integracao 22/24 do dia
int d23=dia[23]=22; // Valor da integracao 23/24 do dia
int d24=dia[24]=1023; // Valor da integracao 24/24 do dia
//-----
// ***** BOTOES posicao e valores analogicos *****
const int SELBTN = 0; // Analog value for the SELECT button
const int LFTBTN = 133; // Analog value for the LEFT button
const int DWNBTN = 273; // Analog value for the DOWN button
const int UPBTN = 460; // Analog value for the UP button
const int RGTBTN = 715; // Analog value for the RIGHT button
//-----
// ***** STRING NO MENU *****
prog_char string_0[] PROGMEM = "1. Contraste " ; //0
prog_char string_1[] PROGMEM = "2. Backlight " ; //1
prog_char string_2[] PROGMEM = "3. Data " ; //2
prog_char string_3[] PROGMEM = "4. Hora " ; //3
prog_char string_4[] PROGMEM = "5. Nivel/Vazao " ; //4
prog_char string_5[] PROGMEM = "6. Reserva A " ; //5
prog_char string_6[] PROGMEM = "7. Reserva B " ; //6
prog_char string_7[] PROGMEM = " Menu Principal " ; //7
prog_char string_8[] PROGMEM = " Contraste " ; //8
prog_char string_9[] PROGMEM = " Backlight " ; //9
prog_char string_10[] PROGMEM = " Data " ; //10
prog_char string_11[] PROGMEM = " Hora " ; //11
prog_char string_12[] PROGMEM = " Nivel e Vazao "; //12
prog_char string_13[] PROGMEM = " Reserva A " ; //13
prog_char string_14[] PROGMEM = " Reserva B "; //14
//-----
//array of pointers to the flash based string table
PGMEM const char *string_table[] = {
    string_0, string_1, string_2, string_3, string_4, string_5, string_6, string_7,
    string_8, string_9, string_10 , string_11 , string_12 , string_13 , string_14
};
//-----
// ***** GLOBAL VARIABLES *****
// LCD
LiquidCrystal_I2C lcd(0x20,16,2); // set the i2c LCD address to 0x20 and define a
16 rows and 2 lines display
byte backlight; // values from 0 to 255 LCD backlight
byte contrast; // values from 0 to 255 LCD contrast
long previousMillis; // will store last time of timeout
unsigned long currentMillis; // will store the current time counter

```



```

boolean BacklightON = true; // used for the control of the backlight timeout
boolean SDCARD = true; // used for disabling the logging in case of SD CARD failure
File myFile;
char DATAFILE[] = "data.log"; // Name of the Data file that will be created on the
SD memomry card, this name must be in the 8.3 DOS format
//-----
// BOTOES
unsigned long buttonLastChecked = 0; // variable to limit the button getting
checked every cycle
short previousval = 0; // initialization of the variable to 0
short pushedbutton = 0; // initialization of the variable to 0
short previousbutton = 0; // initialization of the variable to 0
unsigned long lastbtnpress = 0; // initialization of the variable to 0
//-----
// SENSORES
//-----
const uint8_t trigPin = 22; // 4 Pino D2 do Arduino. (Pino trigger HC-SR04)
const uint8_t echoPin = 24; // 2 Pino D3 do Arduino. (Pino echo HC-SR04)
const uint8_t fluxPin = 26; // 3 Pino D4 do Arduino. (Pino sinal YF-S201)
//-----
// ***** ARDUINO SETUP *****
void setup()
{
//-----
//-----***** SETUP DO LCD e SD CARD e RTC *****
pinMode(LEDPIN, OUTPUT); // sets the LCD LED backlight pin as output
pinMode(CONTRASTPIN, OUTPUT); // sets the LCD Contrast pin as output
pinMode(A1, INPUT); // sets the Analog pin 1 as input
pinMode(A2, INPUT); // sets the Analog pin 2 as input
pinMode(A3, INPUT); // sets the Analog pin 3 as input
Alarm.timerRepeat(15, RepeatTask); // timer that calls the "RepeatTask
function every 15 seconds
Alarm.timerRepeat(1,0,0, RepeatTask); // timer that calls the "RepeatTask
function every hour (up to 6 different timers can be set)
lcd.begin(16, 2); // set up the LCD's number of columns and rows:
lcd.print("T-DATA v1.0"); // Print a message to the LCD.
lcd.setCursor(0, 1); // set the cursor to column 0, line 1 (note: line 1 is
the second line, since counting starts with 0):
lcd.print("Andre Marotta"); // Print a message to the LCD.
backlight=EEPROM.read(1); // Read from the EEPROM the backlight value set by the
interface. It is stored in the EEPROM address 1
analogWrite(LEDPIN, backlight); // Turn On the LCD Backlight to the value stored
in the EEPROM
contrast=EEPROM.read(0); // Read from the EEPROM the contrast value set by the
interface. It is stored in the EEPROM address 0

```

```

    if (contrast > 81) contrast=0; // A formatted or new chip has all it's EEPROM
bytes set to 255, and contrast at 255 is the minimum contrast, so in order to have
something on the display we test new chips here.
    analogWrite(CONTRASTPIN, contrast); // Set the LCD contrast to the value stored
in the EEPROM
#ifdef DEBUG_ON
    Serial.begin(9600); // Serial port initialize
    Serial.println("Data logger BOOT");
#endif
    pinMode(53, OUTPUT); //10 no Uno, 53 no Mega.
    digitalWrite(53, HIGH); // Linha adicionada para o caso do Mega
    if (!SD.begin(SD_CS)) { // SD_CS no caso do Uno
        lcd.clear();
        lcd.print("Cartao SD falhou");
        lcd.setCursor(0, 1); // set the cursor to column 0, line 1 (note: line 1 is
the second line, since counting starts with 0):
        lcd.print(" Log desativado "); // Print a message to the LCD.
        SDCARD = false;
    }
#ifdef DEBUG_ON
    Serial.println("INIT SD Falhou!");
#endif
    Alarm.delay(2000); // For the library TimeAlarms, it is necessary to use the
Alarm.delay function instead of the standard Delay function.
}
    setSyncProvider(RTC.get); // Sync the Arduino clock from the time of the RTC,
resync every 5 minutes (default value)
    Alarm.delay(5000); // Wait for splash screen
    previousMillis = millis(); // Reset timeout counter
    if (SDCARD) {
        myFile = SD.open(DATAFILE, FILE_WRITE); // if SD card present, open the
data.log file, note that only one file can be open at a time, so you have to close
this one before opening another.
        digitalWrite(5);
        myFile.println(";Boot");
        myFile.close(); // close the file
    }
    lcd.clear(); // pretty self explaining, isn't it?
//-----
//-----***** ETHERNET SETUP *****-----
//-----
uint8_t mac[]      = {0x90, 0xA2, 0xDA, 0x00, 0x71, 0x17}; //Endereço MAC
uint8_t ip[]       = {192, 168, 0, 10}; //Endereço IP
uint8_t gateway[]  = {192, 168, 1, 1}; // Gateway
uint8_t subnet[]   = {255, 255, 255, 0}; //Mascara de Subrede

```

```

//-----
-----
Ethernet.begin(mac, ip, subnet); // Inicia porta Ethernet
Udp.begin(localPort);           // Inicia UDP
//pode colocar gateway entre ip e subnet
//Avoid pins 4,10,11,12,13 quando estiver usando ethernet shield
delay(5000);                     // Tempo para abrir o terminal
Serial.begin(9600);              // 115.200 Inicia a porta Serial
//-----
-----
}
// ***** ARDUINO LOOP PRINCIPAL *****
void loop()
{
//-----
-
//PARTE 1 - SENSORES
//-----
-----
uint32_t durationa; // Duração do sinal do sensor de vazao.
uint32_t duration;  // Duração do sinal do sensor ultrassonico.
float cm1;           // Nivel - Sensor ultrasson.
float cm;            // Nivel ( % ).
float vz;            // Vazao - Sensor de vazao.
float vza;           // Vazao - Sensor de vazao.
pinMode(trigPin, OUTPUT); // 1-Habilita o pino D2 como saida digital.
digitalWrite(trigPin, LOW); // 2-Força saida do pino D2 como nivel baixo.
delayMicroseconds(3);      // 3-Espera 3 milisegundos
digitalWrite(trigPin, HIGH); // 4-Inicia o pino D2 como trigger (Pulso
digital).
delayMicroseconds(10);     // 5-Espera 10 milisegundos.
digitalWrite(trigPin, LOW); // 6-Força saida do pino D2 como nivel baixo.
duration = pulseIn(echoPin, HIGH);
cm1=(17*duration)/1000 ;
cm=100-(cm1*10);
durationa = pulseIn(fluxPin, LOW);
if (durationa==0) { // Função if, para corrigir indeterminacao quando
durationa=0.
    vz=0;
    vza=0;
}
else {
    vz=1000000/(durationa*2) ; // vz = 1/ (durationa*2) Convertendo tempo em
frequencia.

```

```

        vza=vz*7,469;                // convertendo frequencia em vazao de acordo com
tabela do fabricante
    }
//-----
//PARTE 2 - MODBUS-TCP
Mb.Run(); //Inicia Biblioteca Modbus
//Mb.R[3]=analogRead(5); // Coloca o valor da entrada analogica 5 no registrador
//a=analogRead(5); // Coloca o valor da entrada analogica 5 em uma variavel
//Serial.print(a); // Imprime na porta serial o valor
//-----

    Mb.R[0]=vza;
    Mb.R[1]=cm;
    Mb.R[2]=12;
    Mb.R[3]=13;
    Mb.R[4]=14;
    Mb.R[5]=15;
//-----
// Modpoll commands
// Read the registers Mb.R[0] ate and Mb.R[6]
// C:/> modpoll -m tcp -t4 -r 1 -c 6 192.168.0.10
//-----

//PARTE 3 - ETHERNET-UDP PROTOCOLO MAROTTA
int packetSize = Udp.parsePacket(); // Variavel que armazena o tamanho do pacote
UDP
if(packetSize)    // Condição: Se houver pacote UDP recebido, execute:
{
    //-----
    //3.1-Identificando origem do pacote UDP recebido
    //-----

    Serial.print("Pacote recebido Tamanho: ");
    Serial.println(packetSize); // Imprime o tamanho do pacote recebido na porta
serial
    Serial.print("De:");
    IPAddress remote = Udp.remoteIP(); // Verifica qual o IP que enviou a pacote
UDP
    for (int i =0; i < 4; i++)
    {
        Serial.print(remote[i], DEC); // Imprime pontos entre os numeros do IP na
porta serial
        if (i < 3)
        {
            Serial.print("."); // Imprime pontos entre os numeros do IP na porta serial
        }
    }
    Serial.print(", Porta: ");

```

```

    Serial.println(Udp.remotePort()); //Imprime o IP de quem enviou o pacote UDP
na porta serial

```

```

//-----
//3.2-Lendo o conteudo do pacote UDP recebido
//-----
Udp.read(packetBuffer,UDP_TX_PACKET_MAX_SIZE);
Serial.println("Conteudo:");
Serial.println(packetBuffer);
//-----

int k1=buffer[1]=vza;           //Ler entrada Analogica 0
int k2=buffer[2]=cm;           //Ler entrada Analogica 1
int k3=buffer[3]=analogRead(2); //Ler entrada Analogica 2
int k4=buffer[4]=analogRead(3); //Ler entrada Analogica 3
int k5=buffer[5]=analogRead(4); //Ler entrada Analogica 4
int k6=buffer[6]=analogRead(5); //Ler entrada Analogica 5
int k7=buffer[7]=digitalRead(2); //Ler pino digital 2 IN-OUT
int k8=buffer[8]=digitalRead(3); //Ler pino Digital 3 IN-OUT-PWM
int k9=buffer[9]=digitalRead(4); //Ler entrada Digital 4 IN-OUT
int k10=buffer[10]=digitalRead(5); //Ler entrada Digital 5 IN-OUT-PWM
int k11=buffer[11]=digitalRead(6); //Ler entrada Digital 6 IN-OUT-PWM
int k12=buffer[12]=digitalRead(7); //Ler entrada Digital 7 IN-OUT
//-----
//-----
//3.3-Executando acao conforme tamanho do pacote UDP recebido (Comandos)
//-----
// packetSize = 1 - Enviar entrada analogica 0
// packetSize = 2 - Enviar entrada analogica 1
// packetSize = 3 - Enviar entrada analogica 2
// packetSize = 4 - Enviar entrada analogica 3
// packetSize = 5 - Enviar entrada analogica 4
// packetSize = 6 - Enviar entrada analogica 5
// packetSize = 7 - Enviar entrada Digital 2
// packetSize = 8 - Enviar entrada Digital 3
// packetSize = 9 - Enviar entrada Digital 4
// packetSize = 10 - Enviar entrada Digital 5
// packetSize = 11 - Enviar entrada Digital 6
// packetSize = 12 - Enviar entrada Digital 7
// packetSize = 13 - Iniciar Massa
// packetSize = 14 - Enviar Massa
//-----
if (packetSize == 1)
{
    Serial.println("Executando Comando 1--LEA1");
    Udp.beginPacket(Udp.remoteIP(), 2002);
    Udp.write(k1);
}

```

```

        Udp.endPacket();
        //delay(2000);
    }
    if (packetSize == 2)
    {
        Serial.println("Ejecutando Comando 2--LEA2");
        Udp.beginPacket(Udp.remoteIP(), 2002);
        Udp.write(k2);
        Udp.endPacket();
        //delay(2000);
    }
    if (packetSize == 3)
    {
        Serial.println("Ejecutando Comando 3--LEA3");
        Udp.beginPacket(Udp.remoteIP(), 2002);
        Udp.write(k3);
        Udp.endPacket();
        //delay(2000);
    }
    if (packetSize == 4)
    {
        Serial.println("Ejecutando Comando 4--LEA4");
        Udp.beginPacket(Udp.remoteIP(), 2002);
        Udp.write(k4);
        Udp.endPacket();
        //delay(2000);
    }
    if (packetSize == 5)
    {
        Serial.println("Ejecutando Comando 5--LEA5");
        Udp.beginPacket(Udp.remoteIP(), 2002);
        Udp.write(k5);
        Udp.endPacket();
        //delay(2000);
    }

    if (packetSize == 6)
    {
        Serial.println("Ejecutando Comando 6--LEA6");
        Udp.beginPacket(Udp.remoteIP(), 2002);
        Serial.println(k6);
        Udp.write(k6);
        Udp.endPacket();
        //delay(2000);
    }
}

```

```

if (packetSize == 7)
{
    Serial.println("Executando Comando 7--LEA7");
    Udp.beginPacket(Udp.remoteIP(), 2002);
    Udp.write(k7);
    Udp.endPacket();
    //delay(2000);
}
if (packetSize == 8)
{
    Serial.println("Executando Comando 8--LEA8");
    Udp.beginPacket(Udp.remoteIP(), 2002);
    Udp.write(k8);
    Udp.endPacket();
    //delay(2000);
}
if (packetSize == 13)
{
    Serial.println("Iniciando Memoria de Massa--");
    inicia=1;
    //delay(2000);
}
if (packetSize == 14)
{
    Serial.println("Executando envio da Memoria de Massa--");
    Udp.beginPacket(Udp.remoteIP(), 2003);
    Udp.write(dia[1]);
    Udp.endPacket();
    Udp.beginPacket(Udp.remoteIP(), 2003);
    Udp.write(dia[2]);
    Udp.endPacket();
    Udp.beginPacket(Udp.remoteIP(), 2003);
    Udp.write(dia[3]);
    Udp.endPacket();
    Udp.beginPacket(Udp.remoteIP(), 2003);
    Udp.write(dia[4]);
    Udp.endPacket();
    Udp.beginPacket(Udp.remoteIP(), 2003);
    Udp.write(dia[5]);
    Udp.endPacket();
    Udp.beginPacket(Udp.remoteIP(), 2003);
    Udp.write(dia[6]);
    Udp.endPacket();
    Udp.beginPacket(Udp.remoteIP(), 2003);
    Udp.write(dia[7]);
}

```

```
Udp.endPacket();
Udp.beginPacket(Udp.remoteIP(), 2003);
Udp.write(dia[8]);
Udp.endPacket();
Udp.beginPacket(Udp.remoteIP(), 2003);
Udp.write(dia[9]);
Udp.endPacket();
Udp.beginPacket(Udp.remoteIP(), 2003);
Udp.write(dia[10]);
Udp.endPacket();
Udp.beginPacket(Udp.remoteIP(), 2003);
Udp.write(dia[11]);
Udp.endPacket();
Udp.beginPacket(Udp.remoteIP(), 2003);
Udp.write(dia[12]);
Udp.endPacket();
Udp.beginPacket(Udp.remoteIP(), 2003);
Udp.write(dia[13]);
Udp.endPacket();
Udp.beginPacket(Udp.remoteIP(), 2003);
Udp.write(dia[14]);
Udp.endPacket();
Udp.beginPacket(Udp.remoteIP(), 2003);
Udp.write(dia[15]);
Udp.endPacket();
Udp.beginPacket(Udp.remoteIP(), 2003);
Udp.write(dia[16]);
Udp.endPacket();
Udp.beginPacket(Udp.remoteIP(), 2003);
Udp.write(dia[17]);
Udp.endPacket();
Udp.beginPacket(Udp.remoteIP(), 2003);
Udp.write(dia[18]);
Udp.endPacket();
Udp.beginPacket(Udp.remoteIP(), 2003);
Udp.write(dia[19]);
Udp.endPacket();
Udp.beginPacket(Udp.remoteIP(), 2003);
Udp.write(dia[20]);
Udp.endPacket();
Udp.beginPacket(Udp.remoteIP(), 2003);
Udp.write(dia[21]);
Udp.endPacket();
Udp.beginPacket(Udp.remoteIP(), 2003);
Udp.write(dia[22]);
```



```

    Udp.endPacket();
    Udp.beginPacket(Udp.remoteIP(), 2003);
    Udp.write(dia[23]);
    Udp.endPacket();
    Udp.beginPacket(Udp.remoteIP(), 2003);
    Udp.write(dia[24]);
    Udp.endPacket();
    //delay(2000);
    Serial.println("Envio da Massa concluido!");
  }

  //-----
  //3.4-Enviando pacote UDP de resposta para o IP que enviou o Pacote
  //-----
  Udp.beginPacket(Udp.remoteIP(), 2004); // Dados para envio para a porta 2002 ou
, Udp.remotePort());
  Serial.println("Resposta:");
  Serial.println(ReplyBuffer);
  Udp.write(ReplyBuffer); // Enviando pacote UDP para local onde foi especificado
acima.
  Udp.endPacket();
  //-----
}

//-----
//PARTE 3 - DATA SHIELD
digitalClockDisplay(); //call function
Alarm.delay(50); // refresh of the alarm trigger, with i2c displays do not choose
values under 50 ms in order to avoid ugly flickering.
if( buttonLastChecked == 0 ) // see if this is the first time checking the
buttons
    buttonLastChecked = millis()+BUTTONDELAY; // force a check this cycle because
we do not need to check every cycle
if( millis() - buttonLastChecked > BUTTONDELAY ) { // make sure a reasonable
delay passed
    if( int buttNum = buttonPushed(BUTTONPIN) ) {
        previousbutton=buttNum;
        mainMenu();
    }
    buttonLastChecked = millis(); // reset the lastChecked value
}
currentMillis = millis();
if(currentMillis - previousMillis > TIMEOUT) {
    if (BacklightON != false) {
        fadeBacklight();
    }
}

```

```

    }

}

//-----
//-----FUNCOES APOS O LOOP PRINCIPAL
//-----
// ***** FUNCTIONS *****
//-----
// ***** LEITURA DE BOTOES *****

int buttonPushed(int BUTTONPIN) {
    int val = ERROR_WINDOW;          // variable to store the read value
    val = analogRead(BUTTONPIN);     // read the input pin
    #ifdef DEBUG_ON
        Serial.println(val); // Output to serial port the analog value of the pressed
button
    #endif
    if (val < (previousval-ERROR_WINDOW) or val > (previousval+ERROR_WINDOW)) {
// Avoid duplicate button presses
        if( val >= (RGTBTN-ERROR_WINDOW) and val <= (RGTBTN+ERROR_WINDOW) ) { // 716
            previousval = val;
            pushedbutton = 4; // RIGHT
            previousMillis = currentMillis; // reset of timeout
            BacklightON = true;
            lastbtnpress = millis(); // reset the lastChecked value
            return pushedbutton;
        }
        else if ( val >= (UPBTN-ERROR_WINDOW) and val <= (UPBTN+ERROR_WINDOW) ) { //
469
            previousval = val;
            pushedbutton = 2; // UP
            previousMillis = currentMillis; // reset of timeout
            BacklightON = true;
            lastbtnpress = millis(); // reset the lastChecked value
            return pushedbutton;
        }
        else if ( val >= (DWNBTN-ERROR_WINDOW) and val <= (DWNBTN+ERROR_WINDOW) ) { //
293
            previousMillis = currentMillis; // reset of timeout
            previousval = val;
            pushedbutton = 3; // DOWN
            BacklightON = true;
            lastbtnpress = millis(); // reset the lastChecked value
            return pushedbutton;
        }
    }
}

```

```

else if ( val >= (LFTBTN-ERROR_WINDOW) and val <= (LFTBTN+ERROR_WINDOW) ) { //
132
    previousMillis = currentMillis;    // reset of timeout
    previousval = val;
    pushedbutton = 1;  // LEFT
    BacklightON = true;
    lastbtnpress = millis(); // reset the lastChecked value
    return pushedbutton;
}
else if( val >= SELBTN and val <= (SELBTN+ERROR_WINDOW) ) {
    previousMillis = currentMillis;    // reset of timeout
    previousval = val;
    pushedbutton = 5; // SELECT
    BacklightON = true;
    lastbtnpress = millis() ; // reset the lastChecked value
    return pushedbutton;
}
else {
    previousval = (-ERROR_WINDOW-1);
    return 0;  // no button found to have been pushed
}
}
else if (pushedbutton < 5) {
    if( millis() - lastbtnpress > BUTTONREPEAT ) { // repeat key presses at
BUTTONREPEAT interval when buttons are keep pressed
        lastbtnpress = millis(); // reset the time of the last key press
        previousMillis = currentMillis;    // reset of timeout
        return pushedbutton;
    }
}
}
//-----
// ***** MENU PRINCIPAL *****
void mainMenu() {
    byte offset=0;
    int buttNum;
    analogWrite(LEDPIN, backlight);
    lcd.clear();
    lcd.noBlink();
    PrintLCD_P(7); //"      Main Menu      " Faz aparecer o nome da string (7) no topo
do Display.
    do {
        PrintLCDAt_P(offset, 0, 1);
        buttNum = buttonPushed(BUTTONPIN);
        previousbutton = buttNum;

```

```

    Alarm.delay(BUTTONDELAY);
    currentMillis = millis();
    switch(buttNum) {
// BOTOES
// Case 1 Botao 1 - Esquerda    - LEFT
// Case 2 Botao 2 - Para cima   - UP
// Case 3 Botao 3 - Para baixo  - DOWN
// Case 4 Botao 4 - Direita     - RIGHT
// Case 5 Botao 5 - Selecciona  - SELECT
        case 1:
            loop();
            break;
        case 2:
            if (offset > 0) offset--;
            break;
        case 3:
            if (offset < 4) offset++; //Ao adicionar elementos no menu, subtrair -1. 4
            itens, offset 3. 5 itens, offset 4.
            break;
        case 5:
            switch (offset) {
                case 0: //Contrast
                    ContrastMenu();
                    break;
                case 1: //Backlight
                    BacklightMenu();
                    break;
                case 2: //Date
                    DateMenu();
                    break;
                case 3: //Hour
                    HourMenu();
                    break;
                case 4: //Analog
                    AnalogMenu();
                    break;
            }
            break;
        }
    }
    while (currentMillis - previousMillis < TIMEOUT);
}

//-----
// ***** SUB-MENUS *****
//-----

```

```
// ***** VAZAO E NIVEL MENU *****
void AnalogMenu() {
//-----
uint32_t durationa; // Duração do sinal do sensor de vazao.
uint32_t duration; // Duração do sinal do sensor ultrassonico.
float cm1;          // Nivel - Sensor ultrasson.
float cm;            // Nivel - ( % )
float vz;            // Vazao - Sensor de vazao.
float vza;           // Vazao - Sensor de vazao.
//-----

    int bar=(contrast/16);
    byte prevcontr = contrast;
    int prevBar = bar;
    lcd.clear();
    PrintLCD_P(12); //"          Analog          "
    do {
//Sequencia de comandos:
pinMode(trigPin, OUTPUT);          // 1-Habilita o pino D2 como saida digital.
digitalWrite(trigPin, LOW);        // 2-Força saida do pino D2 como nivel baixo.
delayMicroseconds(3);              // 3-Espera 3 milisegundos
digitalWrite(trigPin, HIGH);       // 4-Inicia o pino D2 como trigger (Pulso
digital).
delayMicroseconds(10);              // 5-Espera 10 milisegundos.
digitalWrite(trigPin, LOW);        // 6-Força saida do pino D2 como nivel baixo.
duration = pulseIn(echoPin, HIGH);
cm1=(17*duration)/1000 ;
cm=100-(10*cm1) ;
durationa = pulseIn(fluxPin, LOW); //Uno-LOW
if (durationa==0) {                // Função if, para corrigir indeterminacao quando
durationa=0.
    vz=0;
    vza=0;
}
    else {
        vz=1000000/(durationa*2) ; // vz = 1/ (durationa*2) Convertendo tempo em
frequencia.
        vza=vz*7,469;                // convertendo frequencia em vazao de acordo com
tabela do fabricante
    }

    lcd.setCursor(1, 1);             //Posicionar cursor do Display (Coluna, Linha)
    lcd.print(cm); //Imprimir no display Pino D3 do Arduino. (Pino sinal YF-
S201)
    lcd.setCursor(5, 1);
    lcd.print("%");
    lcd.setCursor(9, 1);             //Posicionar cursor do Display (Coluna, Linha)

```

```

    lcd.print(vza); //Imprimir no display Pino D4 do Arduino. (Pino echo HC-
SR04)

    lcd.setCursor(13, 1);
    lcd.print("L/h");
    delay(200);
    //Escrever função a ser executada no menu
    int buttNum = buttonPushed(BUTTONPIN);
    Alarm.delay(BUTTONDELAY);
    currentMillis = millis();
    switch(buttNum) {
    case 1: // esc
        mainMenu();
        break;
    case 2:
        break;
    case 3:
        break;
    case 5: // save
        mainMenu();
        break;
    }
    }
    while (currentMillis - previousMillis < TIMEOUT);
}
//-----
// ***** MENU CONTRASTE *****
void ContrastMenu() {

    int bar=(contrast/16);
    byte prevcontr = contrast;
    int prevBar = bar;
    lcd.clear();
    PrintLCD_P(8); //"          Contrast          "
    do {
        bar=(15-(contrast/5));
        lcd.setCursor(prevBar,1);
        lcd.print(" ");
        lcd.setCursor(bar, 1);
        lcd.print(char(255));
        analogWrite(CONTRASTPIN, contrast);
        int buttNum = buttonPushed(BUTTONPIN);
        Alarm.delay(BUTTONDELAY);
        prevBar = bar;
        currentMillis = millis();
        switch(buttNum) {

```

```

    case 1: // esc
        contrast = prevcontr;
        analogWrite(CONTRASTPIN, contrast);
        mainMenu();
        break;
    case 2:
        if (contrast >=5) contrast=contrast-5; // 0 = contrast MAX
        break;
    case 3:
        if (contrast < 81) contrast=contrast+5; // does not make sense to go upper
        than aprox. 90, the contrast is already at MIN around this value
        break;
    case 5: // save
        EEPROM.write(0, contrast); // write the new contrast value to EEPROM at
address 0
        mainMenu();
        break;
    }
}
while (currentMillis - previousMillis < TIMEOUT);
}
//-----
// ***** MENU BACKLIGHT *****
void BacklightMenu() {
    int bar = (backlight/16);
    byte prevback = backlight;
    int prevBar = bar;
    lcd.clear();
    PrintLCD_P(9); //"    Backlight    "
    do {
        bar=(backlight/16);
        lcd.setCursor(prevBar,1);
        lcd.print(" ");
        lcd.setCursor(bar, 1);
        lcd.print(char(255));
        analogWrite(LEDPIN, backlight);
        int buttNum = buttonPushed(BUTTONPIN);
        Alarm.delay(BUTTONDELAY);
        prevBar = bar;
        currentMillis = millis();
        switch(buttNum) {
            case 1: // ESC
                backlight = prevback;
                analogWrite(LEDPIN, backlight);
                mainMenu();

```

```

        break;
    case 2:
        if (backlight < 240) backlight=backlight+16; // 255 = backlight MAX
        break;
    case 3:
        if (backlight >=16) backlight=backlight-16; // 0 = backlight OFF
        break;
    case 5: // SAVE
        EEPROM.write(1, backlight); // write the new backlight value to EEPROM at
address 1
        mainMenu();
        break;
    }
}
while (currentMillis - previousMillis < TIMEOUT);
}
//-----
// ***** MENU DATA *****
void DateMenu() {
    lcd.clear();
    PrintLCD_P(10); //"    Date    "
    lcd.setCursor(1, 1); //line=1 (0 is the first line), x=3
    printDayOfWeek(weekday());
    lcd.print(" ");
    lcd.print(zerolead(day()));
    lcd.print("/");
    lcd.print(zerolead(month()));
    lcd.print("/");
    lcd.print(year());
    do {
        int buttNum = buttonPushed(BUTTONPIN);
        Alarm.delay(BUTTONDELAY);
        currentMillis = millis();
        switch(buttNum) {
            case 1: // ESC
                mainMenu();
                break;
            case 5: // Call setup menu
                DateSetMenu(); // Back to menu
                break;
        }
    }
    while (currentMillis - previousMillis < TIMEOUT);
}
//-----

```



```
// ***** MENU SETUP DATA *****

void DateSetMenu() {
    byte offset=0;
    int Day,Month,Year;
    Day = day();
    Month = month();
    Year = year();
    lcd.clear();
    PrintLCD_P(7); //"    Date    "
    lcd.setCursor(3, 1); //line=1 (0 is the first line), x=3
    lcd.print(zerolead(day()));
    lcd.print("/");
    lcd.print(zerolead(month()));
    lcd.print("/");
    lcd.print(year());
    lcd.blink();
    do {
        lcd.setCursor(4+(offset*3), 1);
        if (offset == 2) lcd.setCursor(12, 1);
        int buttNum = buttonPushed(BUTTONPIN);
        Alarm.delay(BUTTONDELAY);
        currentMillis = millis();
        switch(buttNum) {
            case 1:
                if (offset > 0) offset--;
                break;

            case 2:
                switch(offset) {
                    case 0:
                        if (Day < 31) Day++;
                        lcd.setCursor(3, 1); //line=2 (1 is the second line), x=5
                        lcd.print(zerolead(Day));
                        break;
                    case 1:
                        if (Month < 12) Month++;
                        lcd.setCursor(6, 1); //line=2 (1 is the second line), x=8
                        lcd.print(zerolead(Month));
                        break;
                    case 2:
                        if (Year < 2099) Year++;
                        lcd.setCursor(9, 1); //line=2 (1 is the second line), x=11
                        lcd.print(Year);
                        break;
                }
            }
    }
}
```

```

        break;

    case 3:
        switch(offset) {
            case 0:
                if (Day > 1) Day--;
                lcd.setCursor(3, 1); //line=2 (1 is the second line), x=5
                lcd.print(zerolead(Day));
                break;
            case 1:
                if (Month > 1) Month--;
                lcd.setCursor(6, 1); //line=2 (1 is the second line), x=8
                lcd.print(zerolead(Month));
                break;
            case 2:
                if (Year > 2010) Year--;
                lcd.setCursor(9, 1); //line=2 (1 is the second line), x=11
                lcd.print(Year);
                break;
        }
        break;

    case 4:
        if (offset < 2) offset++;
        break;

    case 5: // SAVE
        setTime(hour(),minute(),second(),Day,Month,Year); // set Arduino system time
        time_t t = now(); // convert to UNIX type time_t
        RTC.set(t); // set the time of the RTC chip
        lcd.noBlink();
        DateMenu(); // Back to menu
        break;
    }
}

while (currentMillis - previousMillis < TIMEOUT);
lcd.noBlink();
}

//-----
// ***** MENU HORA *****

void HourMenu() {

    lcd.clear();
    PrintLCD_P(11); //"    Hour    "

    do {

```

```

    lcd.setCursor(4, 1); //line=2 (1 is the second line), x=4
    lcd.print(zerolead(hour()));
    lcd.print(":");
    lcd.print(zerolead(minute()));
    lcd.print(":");
    lcd.print(zerolead(second()));
    int buttNum = buttonPushed(BUTTONPIN);
    Alarm.delay(BUTTONDELAY);
    currentMillis = millis();
    switch(buttNum) {
    case 1: // ESC
        mainMenu();
        break;
    case 5: // call setup menu
        HourSetMenu(); // Back to menu
        break;
    }
}

while (currentMillis - previousMillis < TIMEOUT);
}

// ***** Hour Set Menu *****
void HourSetMenu() {

    byte offset=0;
    int Hour,Minute,Second;
    Hour = hour();
    Minute = minute();
    Second = 0;
    lcd.clear();
    PrintLCD_P(8); //"    Hour    "
    lcd.setCursor(4, 1); //line=2 (1 is the second line), x=4
    lcd.print(zerolead(Hour));
    lcd.print(":");
    lcd.print(zerolead(Minute));
    lcd.print(":");
    lcd.print(zerolead(Second));
    lcd.blink();

    do {
        lcd.setCursor(5+(offset*3), 1);
        int buttNum = buttonPushed(BUTTONPIN);
        Alarm.delay(BUTTONDELAY);
        currentMillis = millis();

```

```

switch(buttNum) {
case 1:
    if (offset > 0) offset--;
    break;
case 2:
    if (offset==0) {
        if (Hour < 23) Hour++;
        lcd.setCursor(4, 1); //line=2 (1 is the second line), x=4
        lcd.print(zerolead(Hour));
    }
    else
    {
        if (Minute < 59) Minute++;
        lcd.setCursor(7, 1); //line=2 (1 is the second line), x=4
        lcd.print(zerolead(Minute));
    }
    break;
case 3:
    if (offset==0)
    {
        if (Hour > 0) Hour--;
        lcd.setCursor(4, 1); //line=2 (1 is the second line), x=4
        lcd.print(zerolead(Hour));
    }
    else
    {
        if (Minute > 0) Minute--;
        lcd.setCursor(7, 1); //line=2 (1 is the second line), x=4
        lcd.print(zerolead(Minute));
    }
    break;
case 4:
    if (offset < 1) offset++;
    break;
case 5: // SAVE
    setTime(Hour,Minute,Second,day(),month(),year()); // Set arduino system time
    time_t t = now(); // convert to UNIX type time_t
    RTC.set(t); // set the time of the RTC chip
    lcd.noBlink();
    HourMenu();
    break;
}
}
while (currentMillis - previousMillis < TIMEOUT);

```

```

    lcd.noBlink();
}
//-----
// ***** Print PROGMEM string *****
// This is the gatekeeper function. The only one to really touch the strings in
program memory. This keeps the need
// for a buffer string to only one. So, basically we're only using 17 bytes of RAM
to store strings.
void PrintLCD_P(int which) {
    char buffer[17];
    strcpy_P(buffer, (char*)pgm_read_word(&(string_table[which])));
    lcd.print(buffer);
}
//-----
// ***** Print PROGMEM string at LCD location *****
// Print a string found in program memory to the LCD at a certain location
// Basically this just sets the location and then calls PrintLCD_P to do the work
void PrintLCDat_P(int which, char x, char y) {
    lcd.setCursor(x, y);
    PrintLCD_P(which);
}
//-----
// ***** MOSTRAR DATA / HORA NO DISPLAY LCD *****
void digitalClockDisplay()
{
    // digital clock display of the date and time
    lcd.setCursor(1, 0); //line=1 is the LCD's second line (0 is the first line)
    printDayOfWeek(weekday());
    lcd.print(" ");
    lcd.print(zerolead(day()));
    lcd.print("/");
    lcd.print(zerolead(month()));
    lcd.print("/");
    lcd.print(year());
    lcd.setCursor(4, 1); //line=2 (1 is the second line), x=3
    lcd.print(zerolead(hour()));
    lcd.print(":");
    lcd.print(zerolead(minute()));
    lcd.print(":");
    lcd.print(zerolead(second()));
}
//-----
// ***** ESCREVER DATA / HORA EM ARQUIVO DO SD CARD *****
void digitalClockWrite()
{

```

```

// digital clock display of the date and time
myFile.print(zerolead(day()));
myFile.print("/");
myFile.print(zerolead(month()));
myFile.print("/");
myFile.print(year());
myFile.print(";");
myFile.print(zerolead(hour()));
myFile.print(":");
myFile.print(zerolead(minute()));
myFile.print(":");
myFile.print(zerolead(second()));
}
//-----
// ***** TABELA DIA / NOME DO DIA *****
// ***** The day name corresponding to the day number given by the *****
void printDayOfWeek(int dayOfWeek)
{
    switch(dayOfWeek) {
        case 1:
            lcd.print("Dom");
            break;
        case 2:
            lcd.print("Seg");
            break;
        case 3:
            lcd.print("Ter");
            break;
        case 4:
            lcd.print("Qua");
            break;
        case 5:
            lcd.print("Qui");
            break;
        case 6:
            lcd.print("Sex");
            break;
        case 7:
            lcd.print("Sab");
            break;
    }
}
//-----
// ***** FUNCAO AUXILIAR PARA RELOGIO DIGITAL *****
// ***** Utility function for digital clock/date display: prints leading 0 for

```

```

// single digit numbers *****
//-----
String zeroLead(int digits)
{
    String lead;
    if(digits < 10) {
        lead = '0'+String(digits);
    }
    else {
        lead = String(digits);
    }
    return lead;
}
//-----
//***** TASK *****
// ***** Tasks to be executed at the interval defined in the Timer *****
void RepeatTask () {
    if (SDCARD) {
        myFile = SD.open(DATAFILE, FILE_WRITE); // if SD card present, open the
        data.log file, note that only one file can be open at a time, so you have to close
        this one before opening another.
        //-----
        uint32_t durationa; // Duração do sinal do sensor de vazao.
        uint32_t duration; // Duração do sinal do sensor ultrassônico.
        float cm;          // Nivel - Sensor ultrasson.
        float vz;          // Vazao - Sensor de vazao.
        float vza;         // Vazao - Sensor de vazao.
        pinMode(trigPin, OUTPUT);          // 1-Habilita o pino D2 como saída digital.
        digitalWrite(trigPin, LOW);        // 2-Força saída do pino D2 como nível baixo.
        delayMicroseconds(3);              // 3-Espera 3 milisegundos
        digitalWrite(trigPin, HIGH);       // 4-Inicia o pino D2 como trigger (Pulso
        digital).
        delayMicroseconds(10);             // 5-Espera 10 milisegundos.
        digitalWrite(trigPin, LOW);        // 6-Força saída do pino D2 como nível baixo.
        duration = pulseIn(echoPin, HIGH);
        cm = (17*duration)/1000 ;
        durationa = pulseIn(fluxPin, LOW);
        if (durationa==0) {                // Função if, para corrigir indeterminação quando
        durationa=0.
            vz=0;
            vza=0;
        }
        else {
            vz=1000000/(durationa*2) ; // vz = 1/ (durationa*2) Convertendo tempo em
            frequência.

```

```

        vza=vz*7,469;                // convertendo frequencia em vazao de acordo com
tabela do fabricante
    }

    digitalWriteWrite();
    myFile.print(";");
    myFile.print(cm);
    myFile.print("-cm");
    myFile.print(";");
    myFile.print(vza);
    myFile.print("-L/h");
    myFile.println(";"); //PRINT-LN: finaliza linha
    myFile.close(); // close the file
    }
}

//-----
// ***** DEGRADE DA LUZ DE FUNDO *****
// ***** Fade backlight after inactivity timeout and reset to date/time
default display *****
void fadeBacklight()
{
    BacklightON = false;
    // fade out from max to min in increments of 1 point:
    lcd.clear();

    for(int fadeValue = backlight; fadeValue >= 0; fadeValue --=1) {
        // Dims the backlight from the current value to 0 (OFF)
        analogWrite(LEDPIN, fadeValue);
        Alarm.delay(10); // wait for 10 milliseconds to see the dimming effect.

        if (millis()-currentMillis > 50) { // To avoid nasty flickering of the display,
do not refresh it on every cycle, keep at least 50 mS between each display refresh
            digitalWriteDisplay();
            currentMillis = millis();
        }

        // Check if some button was pressed during the dimming progress
        if( int buttNum = buttonPushed(BUTTONPIN) ) {
            analogWrite(LEDPIN, backlight);
            currentMillis = millis(); // reset of timeout
            BacklightON = true;
            mainMenu();
            break;
        }
    }
}
}

```


REFERÊNCIAS

ALBERTO, José. **Planejamento da capacidade de produção** Disponível em: <<http://portal.faculadadedeilheus.com.br/Documentos/SERGIO%20DAVID%20FERREIRA%20CRUZ/ADM%20ADM%20PRODUÇÃO%20I%20S05%202012.1/A%20-%20CESUPI%20-%202012-1%20-%20ADM%20PROD%20I%20-%206o%20Sem%20-%207o%20Enc%20-%202009-03-12.pdf>>. Acesso em: nov. 2014.

APUANO, Francisco Gabriel, Idoeta Ivan V. **Elementos de eletrônica digital**. São Paulo, Érica, 2007.

ARDUINO. Disponível em: <<http://www.arduino.cc>>. Acesso em: nov. 2014.

BARROS. A., Erick Schulz Soares. **Gerenciamento de Energia: ações administrativas e técnicas de uso adequado da Energia elétrica**. São Paulo, Érica 2010.

BAXTER, M., **Projeto de Produto: Guia Prático para o Desenvolvimento de Novos Produtos**. São Paulo - SP, Brasil: Editora Edgard Blücher Ltda, 1ª Edição, 1998. ISBN 852120150-8.

CAETANO. A., Erick Schulz Soares. **Projeto de um medidor ultrassônico de nível e medidor de turbidez**. Viçosa, UFV 2013.

CNI Confederação Nacional da Indústria. Disponível em: <<http://www.portaldaindustria.com.br/cni/imprensa/2013/05/1,22994/investimentos-em-automacao-potencializam-competitividade-da-industria.html>>. Acesso em: nov. 2014.

DUNN, Willian C. **Fundamentos de instrumentação industrial e controle de processos**. São Paulo, Bookman, 2013.

EBAH: Medição de nível. Disponível em <http://www.ebah.com.br/content/ABAAAe5CAAF/medicao-nivel> Acesso em Out. 2015.

ELEC FREAKS, Ultrasonic Raring Module HC – SR04. Disponível em: <<http://www.elec Freaks.com>>. Acesso em: nov. 2014.

HALLYDAY, David, Krane Kenneth S., Resnick, Robert. **Física, Volume 2**. São Paulo, LTC, 2003.

IDOETA, Ivan Valeije, Capuano, Francisco Gabriel. **Elementos de eletrônica digital**. São Paulo, Érica, 2008.

INCONTROL. Disponível em: <<http://www.incontrol.com.br>>. Acesso em: nov. 2014.

INSTRUMATIC: Disponível em < www.instrumatic.com.br > Acesso em Out. 2015.

LABORATORIO DE GARAGEM. Disponível em: <<http://www.labdegaragem.com> >. Acesso em: nov. 2014.

LUGLI, Alexandre Baratella, Santos, Max Mauro Dias. **Sistemas Fieldbus para automação industrial**. São Paulo, Érica, 2013.

LUGLI, Alexandre Baratella, Santos, Max Mauro Dias. **Redes industriais para Automação industrial**. São Paulo, Érica, 2014.

MACIEL, Braian Konzgen¹, SOARES, Luciane Baldassari, AGUIAR, Ygor

Quadros. **Barramento Serial SPI e I2C**. Disponível m:

<<http://www.ebah.com.br/content/ABAAAE7xoAL/barramento-serial-spi-i-c>>. Acesso em: Ago. 2015

MODBUS: The Modbus organization. Disponível em: < <http://www.smodbus.org> > Acesso em: nov. 2014.

MONK Simon. **30 projetos com Arduino**. Porto Alegre, Bookman, 2014.

MORSE, I. E. e TSE, F. S., **Measurement and Instrumentation in Engineering**, New York: Marcel Dekker, Inc., 1989.

NERYS, José Wilson Lima, **Microprocessadores e microcontroladores**. UFG 2010.

OLIVEIRA, Amauri. **Apresentação – Escoamento de fluido**, Salvador, UFBA 2015.

OSHWA Open source Hardware Association. Disponível em: < <http://www.oswa.org> >. Acesso em: nov. 2014.

OSI Open source Initiative. 1998. Disponível em: < <http://www.opensource.org> >. Acesso em: nov. 2014.

SERPRO: Ministério do planejamento avança rumo à implantação do software livre num plano que já é referência para outros órgãos do governo. Disponível em: <http://www4.serpro.gov.br/imprensa/publicacoes/tema-1/antigas%20temas/tema_179/materiais/ministerio-do-planejamento>. Acesso em: ago 2015.

STREET, Victor. **Mecânica dos Fluidos**. São Paulo : McGraw-Hill do Brasil, 1969.

TCP/IP: TCP/IP Guide. Disponível em: < <http://www.tcpipguide.com> > Acesso em: nov. 2014.

TETRA Mundo Tetra: Tudo sobre a tecnologia Tetra no Brasil e no mundo..Disponível em: < <http://www.mundotetra.com/2011/08/04/porque-evitar-solucoes-proprietarias/> >. Acesso em: nov. 2014.

WEBSTER, John G., **The Measurement, Instrumentation and Sensors handbook**, São Paulo, CRCCPRESS, 1999.

WIKIBOOKS: Mecânica dos fluidos. Disponível em:
<https://pt.wikibooks.org/wiki/Mecânica_dos_fluidos/O_tubo_de_Pitot>. Recuperado em out. 2015.

WIKIPEDIA: Arduino. Disponível em: < <https://pt.wikipedia.org/wiki/Arduino>>. Recuperado em out.2015.

YIFA The Pastics LTd, Product Introdction sensor YF – 201. Disponível em:
< <http://www.yifa.com> >. Acesso em: nov. 2014.