UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE ENGENHARIA ELÉTRICA
ENGENHARIA ELÉTRICA

FERNANDO CLEMENTINO VIGANÓ

DATALOGGER PARA DADOS METEOROLÓGICOS

TRABALHO DE CONCLUSÃO DE CURSO

CORNÉLIO PROCÓPIO

2016

FERNANDO CLEMENTINO VIGANÓ

DATALOGGER PARA DADOS METEOROLÓGICOS

Trabalho de Conclusão de Curso do curso de Engenharia Elétrica da Universidade Tecnológica Federal do Paraná - UTFPR como requisito parcial para a obtenção do título de Bacharel em Engenharia Elétrica

Orientador: Prof. Dr. Kleber Romero Felizardo

CORNÉLIO PROCÓPIO

2016

Universidade Tecnológica Federal do Paraná
Campus Cornélio Procópio
Departamento de Engenharia Elétrica
Curso de Engenharia Elétrica

FOLHA DE APROVAÇÃO

Fernando Clementino Viganó

Datalogger para dados meteorológicos

Trabalho de conclusão de curso apresentado às 15:50hs do dia 09/06/2016 como requisito parcial para a obtenção do título de Engenheiro Eletricista no programa de Graduação em Engenharia Elétrica da Universidade Tecnológica Federal do Paraná. O candidato foi arguido pela Banca Avaliadora composta pelos professores abaixo assinados. Após deliberação, a Banca Avaliadora considerou o trabalho aprovado.

_____

Prof(a). Dr(a). Kleber Romero Felizardo - Presidente (Orientador)

_____

Prof(a). Dr(a). Francisco de Assis Scannavino Junior - (Membro)

_____

Prof(a). Dr(a). Wagner Endo - (Membro)

AGRADECIMENTOS

Aos meus familiares pelos valiosos incentivos aos estudos.

Aos professores e colegas do curso, pela troca de experiências.

A todos que direta ou indiretamente colaboraram na execução deste trabalho.

Em especial a minha mãe, pelo grande esforço para manter meus estudos.

RESUMO

VIGANÓ, Fernando Clementino. Datalogger para dados metereológicos. 2016. 82f. Trabalho de Conclusão de Curso (Graduação) - Curso Superior de Engenharia Elétrica. Universidade Tecnológica Federal do Paraná, Cornélio Procópio, 2016.

Este trabalho apresenta o desenvolvimento de um datalogger utilizado para registrar dados metereológicos para a agricultura. Entre esses dados estão intensidade de luminosidade, temperatura do ambiente, umidade relativa do ar e umidade relativa do solo.

Palavras-chave: datalogger, sistema embarcado, Arduino, agricultura.

ABSTRACT

This work presents the development of a project to register meteorlogical data for the agriculture. Some of the data are luminosity intensity, environmental temperature, air relative humidity and soil relative humidity. It presents the types of sensors, the embedded systems with their peripherals as well as types of communication between them. At the end, the functioning of the project is showed through the figures obtained from the sensors.

Keywords: datologger, embedded system, Arduino, agriculture.

SUMÁRIO

1 INTRODUÇÃO

O clima tem influência na agricultura por afetar os processos metabólicos das plantas relacionados a produção vegetal. Segundo (HOOGENBOOM, 2000) entre os fatores que influenciam na cultura, tem-se a chuva, a temperatura do ar, a umidade do ar e do solo, a radiação solar e o vento. Sendo assim, o uso de informações meteorológicas e climáticas é fundamental para a agricultura se tornar uma atividade sustentável.

Um dos problemas agrícolas é quando a plantação sofre variações climáticas prejudiciais e o agricultor não tem essas informações necessárias para lidar com a situação. A luminosidade influência diretamente no desenvolvimento da planta, pois a planta que tem maior capacidade de captar a luz consegue melhorar seu sistema radicular, ampliar suas raízes e absorver água e nutrientes (KHATOUNIAN,2003). Em relação a temperatura, as raízes são sensíveis ao aquecimento, pois a variação atrapalha na absorção de nutrientes. A água é responsável pelas reações vitais, pelo transporte interno da seiva bruta, seiva elaborada e também é a matéria prima para a fotossíntese (KHATOUNIAN,2003).

Para Mavi e Tupper (2004) as informações agrometeorológicas podem ser utilizadas para o planejamento dos cultivos. Além disso, essas informações podem ser empregadas no processo de tomada de decisão, auxiliando o agricultor quanto ao melhor momento para determinadas práticas agrícolas.

Segundo SENTELHAS; MONTEIRO, 2009 essas informações são caracterizadas em três graus de complexidade. As informações de primeiro grau são geralmente numéricas, como os dados meteorológicos puros ou derivados de cálculos simples, como por exemplo, o balanço hídrico do solo, que mede a quantidade de água disponível. As de complexidade de segundo grau se caracterizam por dados meteorológicos e parâmetros específicos da agricultura, mostrando a reação da cultura ao clima. As informações de terceiro grau requer a participação de profissionais capacitados pois indicam as ações de manejo mais adequadas para o estado no qual a cultura se encontra em dado momento.

A cultura de precisão utiliza tecnologia de informação baseada no princípio de variabilidade de solo e clima. Uma das aplicações do datalogger é fornecer informações para apontar possíveis causas climáticas que podem limitar a produção da cultura de precisão. Desta forma, os dados adquiridos servem para orientar o produtor sobre as condições climáticas de sua vegetação.

Neste trabalho será desenvolvido um equipamento (datalogger) capaz de obter informações agrometereológicas e armazená-las para posteriormente tratar estas informações através de um computador. Tais informações são baseadas nas leituras dos sensores de temperatura, de umidade do ar, de umidade do solo e de radiação ultravioleta.

A justificativa deste trabalho deve-se ao fato da agricultura ser o setor econômico que mais gasta água no Brasil. Tendo isso em vista, torna-se um fator muito relevante o desenvolvimento de um dispositivo capaz de monitorar dados relacionados a vegetação com o intuito de auxiliar nas tomadas de decisão no que se refere a economia de água.

1.2 OBJETIVOS

Este trabalho tem como objetivo desenvolver um datalogger para monitoramento e armazenamento de dados agrometereológicos. Este datalogger será baseado na placa Arduino MEGA e irá apresentar os seguintes itens:

- Interface com o operador através de botões e display LCD para configurações iniciais requeridas pelo sistema, tais como data, hora e intervalo de tempo para leitura dos dados.
- Leitura dos dados obtidos de sensores de temperatura, de umidade do ar, de umidade do solo e de radiação ultravioleta em intervalos de tempo pré-programados pelo usuário.
- Armazenamento dos dados de leitura dos sensores contendo também a data e a hora das medições em um cartão de memória SD.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 DATALOGGER

Um datalogger é um dispositivo eletrônico que tem a capacidade de registrar dados ao longo do tempo previamente determinado. Os dados variam de acordo com o tipo de monitoramento desejado pelo usuário, pois dependem dos sensores ligados ao dispositivo. Os registros ficam armazenados em memória e podem ser acessados posteriormente em um computador, quando desejar.

2.1.1 Tipos de Dataloggers

O datalogger da figura 1 registra dados de temperatura e umidade do ambiente através de sensores de boa precisão e mostra no display a variação máxima e mínima ocorrida durante a aquisição de dados. Possui 2 canais de saída que podem ser utilizados na irrigação. A configuração deste datalogger é feita através de um software via comunicação USB.

Figura 1 - Registrador Eletrônico de Umidade e Temperatura.

Fonte: Adaptado site (www.shop.bb-sensors.com).

Um outro exemplo de dispositivo no mercado é o datalogger de 7 canais (figura 2.) aplicado a ciências do solo, agronomia, horticultura e agricultura. Possui dois sensores de umidade do solo, dois sensores de temperatura, um pluviômetro e um medidor de fluxo para avaliar a eficiência na irrigação, qualidade e produção vegetal.

Figura 2 - Coletor de dados e controlador de irrigação.

Fonte: Adaptado site (/www.delta-t.co.uk).

O datalogger da figura 3 é um equipamento que se diferencia pela sua robustez e resistência à água. Seu sistema programável suporta até 60 canais com programação independente, ou seja, cada canal pode ser programado para cada tipo de sensor e taxa de amostragem desejada.

Figura 3 – Coletor de dados resistente a água

Fonte: Adaptado site (/www.delta-t.co.uk).

A figura 4 apresenta um datalogger utilizado para medir a tensão hídrica do solo ou substrato junto às raízes da planta em áreas de campo de golfe. O sistema tem a capacidade de monitorar 200 m² com seus dois ramais. Um ramal controla sensores ligados ás raízes com 15 cm de profundidade e o outro ramal controla sensores ligados ás raízes com 50 cm de profundidade.

Figura 4- Modelo de datalogger para campos de golfe.

Fonte: Adaptado site (www.shop.bb-sensors.com).

2.2 ARDUINO

O Arduino é uma plataforma composta por um microcontrolador Atmel AVR de 8 bits, um cristal oscilador de 16MHz, um regulador de tensão de 5V, um botão de reset e uma porta USB utilizada tanto para gravação do microcontrolador tanto para comunicação com um computador (MCROBERTS, 2015).

Cada modelo de Arduino possui pinos de entradas e saídas analógicas e digitais para interagir com dispositivos externos, tais como: leds, sensores, teclados, displays, módulos, dentre outros. A plataforma Arduino é bastante versátil e popular devido aos shields, que são placas de circuito que são encaixadas perfeitamente por cima do Arduino com o intuito de expandir sua capacidade. O Arduino Ethernet Shield, por exemplo, permite conectar o arduino a uma rede local. O Arduino WiFi Shield faz o mesmo que o Ethernet Shiled mas sem a necessidade de um cabo de rede.

O Arduino possui um software gratuito conhecido como Arduino IDE como ambiente de programação e gravação. A linguagem de programação é parecida com a linguagem C/C++ (MCROBERTS, 2015).

2.2.1 Arduino Mega

O Arduino Mega é uma plataforma de prototipagem open-source de hardware baseada no microcontrolador ATmega 2560 que possui um processador de 16 MHZ com 54 pinos de entrada ou saída digital, dos quais 15 podem ser usados como PWM. Possui 16 entradas analógicas de 10 bits, 128 KB de memória flash, 8KB de RAM, 4KB de EEPROM, conexão USB, canais para comunicação serial através dos pinos RX e TX e canais para comunicação SPI e I2C (SIMON, 2013).

Figura 5 – Arduino Mega.

Fonte: Adaptado de McRoberts (2015).

2.2.2 Modo de Baixo Consumo

Para maior autonomia da bateria (ou pilhas) que alimenta o circuito é importante fazer com que o microcontrolador opere em modo de baixo consumo de energia. O arduino pode ser configurado para trabalhar neste modo através dos comandos “sleep_cpu” e “sleep_disable”. É possível escolher quais funções serão habilitadas e quais irão ficar desabilitadas quando o Arduino entra em modo sleep (SIMON,2015). A tabela 1 mostra os comandos utilizados pelo Arduino neste modo de operação e quais os periféricos do microcontrolador podem ser desabilitados neste modo.

Tabela1- Funções para baixo consumo

Função	Descrição
power_adc_disable	Desabilita as entradas analógicas
power_spi_disable	Desabilita a interface SPI
power_twi_disable	Desabilita TWI(I2C)
power_usart0_disable	Desabilita a UART serial
power_timer0_disable	Desabilita o Temporizador 0
power_timer1_disable	Desabilita o Temporizador 1
power_timer2_disable	Desabilita o Temporizador 2
power_all_disable	Desabilita todos os módulos listados acima

Fonte: Adaptado Monk (2015)

Algumas das principais baterias (Figura 6) utilizadas na alimentação do Arduino são: LiPo de 3,7 V de 850mAh ou 2.400mAh, bateria de NiMh de 9V de 565mAh e bateria de lítio de 3V (SIMON,2015).

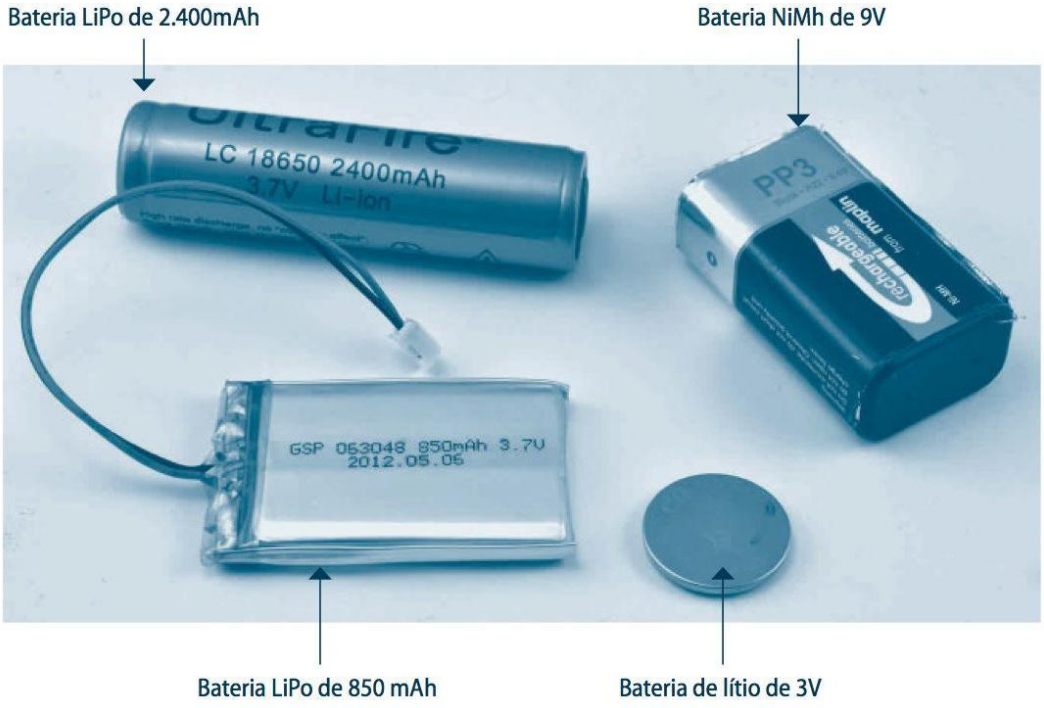

Figura 6- Baterias para alimentar placas de Arduino.

Fonte: Monk (2015).

2.3 SENSORES

Segundo Thomazini (2013) um sensor analógico pode assumir qualquer valor no seu sinal de saída ao longo do tempo desde que esteja dentro da sua faixa de operação. Entre as grandezas físicas que podem assumir esses valores tem-se: temperatura, umidade, luminosidade, dentre outros.

2.3.1 Sensores de Temperatura

Um exemplo de sensor de temperatura são os termistores (figura 7), que são resistores sensíveis a pequenas variações de temperatura. Seus componentes resistivos são compostos de óxidos de metais como manganês, níquel, cobalto, cobre, ferro e titânio (THOMAZINI,2013).

Os termistores são classificados em dois tipos: coeficiente positivo de temperatura (PTC), em que a resistência aumenta com o aumento da temperatura; e coeficiente negativo de temperatura (NTC), na qual a resistência diminui com o aumento da temperatura (THOMAZINI,2013).

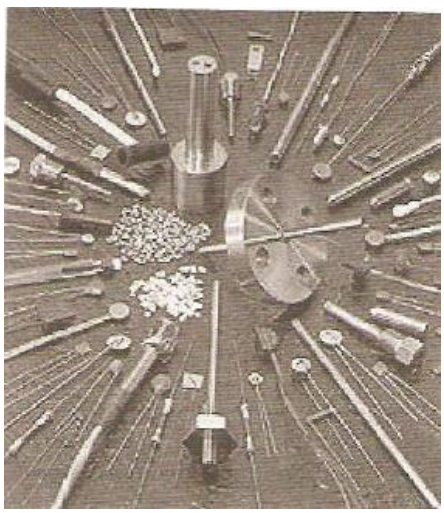

Figura 7 – Modelos de termistores.

Fonte: Thomazini (2013).

Segundo THOMAZINI,2013, o tipo NTC é o mais usual, porém a relação entre resistência e temperatura não é linear, sendo representada pela equação de Steinhart & Hart:

$$\frac{1}{T} = a + bln(R) + cln^3(R). \quad (1)$$

Os valores de a, b e c da equação 1 são coeficientes que dependem do material usado, R é o valor da resistência e T é o valor da temperatura.

Outro exemplo de sensor de temperatura é o termopar, descoberto em 1.821 por T.J.Seebeck. Em um circuito fechado, com fios feito de dois metais heterogêneos, Seebeck observou que flui uma corrente se a temperatura de uma junção T1 estiver acima da temperatura T2 como mostra a figura 8(THOMAZINI,2013).

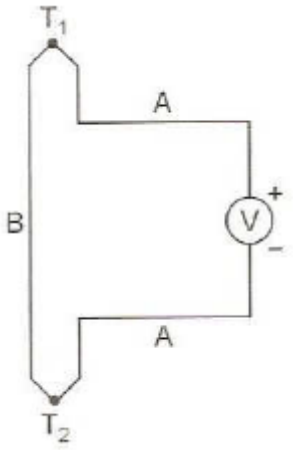

Figura 8 – Circuito utilizado por Seebeck.

Fonte: Thomazini (2013).

A força eletro-motriz gerada no circuito é explicada por três efeitos, sendo eles: efeito de Thomson, onde há criação de campo elétrico devido ao aquecimento de uma barra, efeito de Seebeck onde há uma circulação de corrente no circuito quando existe dois metais de naturezas diferentes ocorrendo uma diferença de temperatura entre as junções, e por último efeito de Peltier que consiste na liberação ou absorção de calor em uma junção termoelétrica (THOMAZINI,2013).

Para medições a longa distância, existem as termorresistências, constituídas por um filamento delgado de metal feito de platina ou níquel, onde a resistência varia com a temperatura. Esse tipo de sensor de temperatura geralmente é utilizado na indústria (THOMAZINI,2013).

2.3.1.1 Sensores Térmicos Eletrônicos

Os sensores térmicos eletrônicos são utilizados na montagem em placas de circuito impresso. Como exemplo tem-se os diodos, transistores e os circuitos integrados.

O diodo comum é constituído por material de silício. Quando polarizado diretamente com uma corrente de 1mA, o diodo tem uma queda de tensão próxima de 0,62V a 25 °C. Essa tensão decresce 2mV para cada °C de aumento na temperatura (THOMAZINI,2013).

Um outro exemplo de sensor de temperatura eletrônico são os transistores, feitos de material de silício. O modelo mais comum é o 2N2222, pois as suas curvas de polarização e o seu encapsulamento metálico facilitam seu uso (THOMAZINI,2013).

O sensor de temperatura LM35 é um exemplo de sensor baseado em circuito integrado e que oferece alta precisão por conter circuitos linearizados. Opera na faixa de -55 a 150°C. Seu fator de escala linear é de +10 mV/°C. Outro exemplo é o sensor de temperatura LM75 que possui interface de comunicação I²C, com precisão térmica de ±2° C e opera na faixa de temperatura entre -55° C a +125° C. (THOMAZINI,2013).

Outra forma de medir a temperatura é através dos pirômetros. São sensores que utilizam a radiação de um corpo sem a necessidade de contato. O problema é que este tipo de sensor pode apresentar erro de medição em determinadas situações de operação. Por exemplo, para medições da radiação térmica em instalações ao ar livre, a radiação solar provoca indicações de temperaturas mais altas. Para eliminar esse tipo de erro, existe um filtro ótico com sensibilidade espectral acima de λ=7,7µm que elimina a participação solar e outros radiadores de infravermelho, possibilitando a medida correta (THOMAZINI,2013).

O sensor de temperatura termopilha (figura 9) permite a leitura de temperatura a distância e com baixo custo. É formado por vários termopares conectados em série tendo acuracidade de ±1°C e precisão de 0,1°C (THOMAZINI,2013).

Figura 9 – Dispositivos encapsulados de termopilhas.
Fonte: Thomazini (2013).

2.3.2 Sensores de Umidade

A água é encontrada na natureza em três estados físicos: líquido, sólido ou gasoso (vapor de água). A umidade absoluta é a quantidade real de água contida no ar. Ela muda com a temperatura do ar, sendo expressa em gramas por metro cúbico de ar. A umidade relativa é o quociente entre a quantidade de umidade presente a uma determinada temperatura e a máxima quantidade de vapor de água contida no ar nessa temperatura (THOMAZINI,2013).

O ar é considerado saturado quando a atmosfera contém todo vapor de água que pode absorver. Esse ponto de saturação é chamado de ponto de orvalho. A tabela 2 mostra a relação da temperatura com a quantidade de vapor de água que se encontra no ponto de orvalho (THOMAZINI,2013).

Tabela 2 – Concentração de vapor de água em relação a temperatura.

Temperatura	Concentração de Vapor de água em gramas por metros cúbicos de ar
0ºC	4,8
10ºC	9,3
20ºC	17,7
25ºC	22,8
30ºC	30,0
35ºC	39,2

Fonte: Thomazini (2013).

Para a escolha do sensor de umidade deve-se respeitar sua faixa de operação, pois em algumas aplicações os sensores operam 90% do tempo em uma faixa ou região muito estreita; e 10% do tempo em uma região bem distante da faixa normal, sendo necessário neste caso a utilização de dois sensores, cada um funcionando na região de operação na qual se aplica (THOMAZINI,2013).

A manutenção dos sensores de umidade deve ser rotineira e o tipo de manutenção vai depender do equipamento. Por exemplo, sensores de óxido de alumínio quando contaminados perdem a calibração e devem ser calibrados novamente. Já outros sensores de umidade, como os de células de ponto de orvalho de cloreto de lítio e os de células eletrolíticas de pentóxido de fósforo; não perdem facilmente sua calibragem. Os sensores de umidade podem durar de alguns meses até 20 anos. Os sensores que são considerados com vida curta são os sensores mais econômicos como os de células de ponto de orvalho, elementos de UR, óxido de alumínio, dentre outros. Já os que são considerados com vida longa tem-se como exemplo os sensores do tipo condensação devido a sua construção inerte (THOMAZINI,2013).

2.3.2.1 Princípios de medição

O princípio de medição do sensor de umidade se baseia no princípio capacitivo de um filme fino composto de uma lâmina de tântalo e outra de cromo, tendo como dielétrico um polímero. A capacidade varia de acordo com a umidade relativa do ambiente e o instrumento faz a conversão eletronicamente indicando a umidade relativa no display (THOMAZINI,2013).

2.3.2.2 Sensores Elétricos de Umidade Relativa

Os sensores elétricos de umidade possuem tempo de resposta rápido quando comparado com o tempo de resposta dos processos industriais, são pequenos, debaixo custo, de boa precisão e são utilizados em locais de baixa contaminação (uma vez contaminado não é possível fazer o reparo) (THOMAZINI,2013).

2.3.2.3 Sensor de Óxido de Alumínio

Este sensor consiste em um capacitor com eletrodos de alumínio e ouro separados por óxido de alumínio e são usados em aplicações petroquímicas em que baixos pontos de orvalhos são controlados e precisões mais baixas são aceitáveis. Este tipo de sensor possui ampla faixa de medição, são pequenos e podem medir pontos de orvalho de até -75°C (THOMAZINI,2013).

2.3.2.4 Higrômetro Eletrolítico

O higrômetro eletrolítico é utilizado para controlar níveis de vapor de água em gases limpos, inertes e secos. O higrômetro eletrolítico utiliza uma célula revestida por uma fina camada de pentóxido fosfórico, e pode ser aplicado na maioria dos gases elementares e inertes e a compostos orgânicos e inorgânicos que não reagem com o pentóxido fosfórico. Este sensor não necessita de calibragem, a célula eletrolítica tem

boa precisão em ampla faixa de medição, não apresenta histerese (THOMAZINI,2013).

2.3.2.5 Higrômetro Óptico de Condensação

O higrômetro óptico de condensação é considerado preciso, seguro e possui uma ampla faixa de medição de ponto de orvalho. O higrômetro está sujeito a contaminação por materiais que não seja a água e sua manutenção é simples, pois a maioria usa câmeras de condensação que podem ser abertas permitindo a limpeza do espelho com um pano. Esse instrumento pode ser aplicado em locais onde é necessária máxima precisão na determinação de teor de vapor de água (THOMAZINI,2013).

As faixas de medição úteis da maioria dos principais sensores estão resumidas na figura 10.

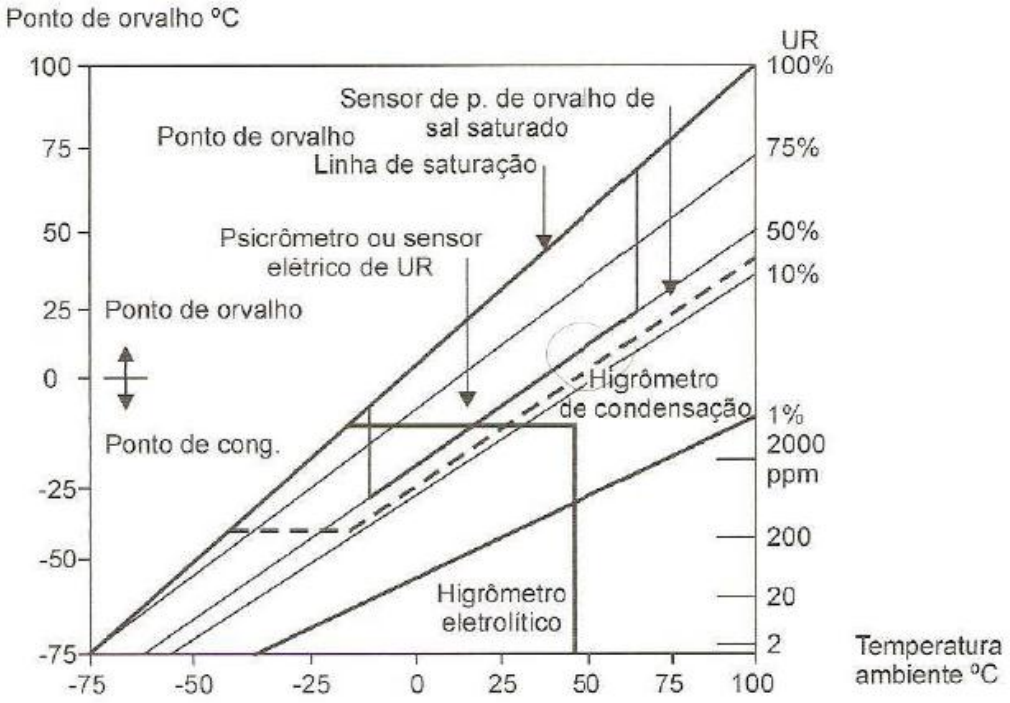

Figura 10 – Faixas de medição úteis de vários sensores.

Fonte: Thomazini (2013).

2.4 PROTOCOLO DE COMUNICAÇÃO SPI

O protocolo de comunicação SPI é um padrão de barramento serial que pode ser utilizado para conectar periféricos do Arduino. Este protocolo utiliza quatro pinos de comunicação, sendo eles: SCLK (*System Clock*), MOSI (*Master Out Slave In*), MISO (*Master In Slave Out*) e SS (Save Select) (MONK,2015).

Os pinos MOSI e MISO são responsáveis pela transferência de dados entre os dispositivos mestre e escravo. O pino SCLK faz a sincronização entre os dispositivos mestre e escravo. O pino SS seleciona o dispositivo escravo que irá se comunicar com o dispositivo mestre e também pode encerrar a execução dos comandos transmitidos pelo dispositivo mestre (MONK,2015). A figura 11 mostra um exemplo de comunicação SPI entre o mestre (Arduino) e dois dispositivos escravos.

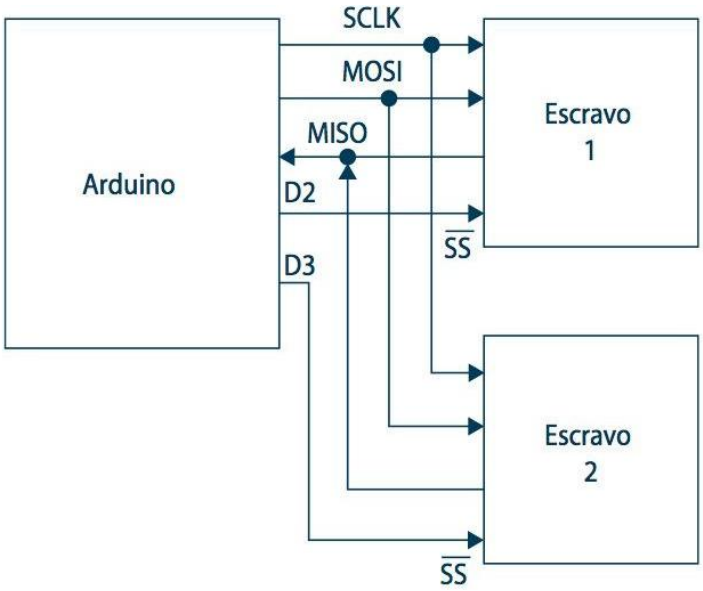

Figura 11– Comunicação SPI entre o Arduino (mestre) e dois dois dispositivos escravos. Fonte: Monk (2015).

A tabela 3 mostra os números correspondentes dos pinos SCLK, MOSI e MISO para diversas versões de Arduino. O pino SS pode ser qualquer pino de saída digital.

Tabela 3 – Conexões SPI nas placas de Arduino

Placa	SCLK	MOSI	MISO
Uno	13 (ICSP3)	11 (ICSP4)	12 (ICSP1)
Leonardo	ICSP3	ICSP4	ICSP1
Mega2560	52 (ICSP3)	51(ICSP4)	50(ICSP1)
Due	ICSP3	ICSP4	ICSP1

Fonte: Monk (2015).

2.5 COMUNICAÇÃO I2C

A comunicação I2C é um protocolo de comunicação serial que utiliza dois pinos de dados, onde o microcontrolador atua como mestre e cada um dos escravos tem um endereço exclusivo para identificar o dispositivo no barramento (MONK,2015). A figura 12 mostra alguns dispositivos I2C que podem ser usados no Arduino.

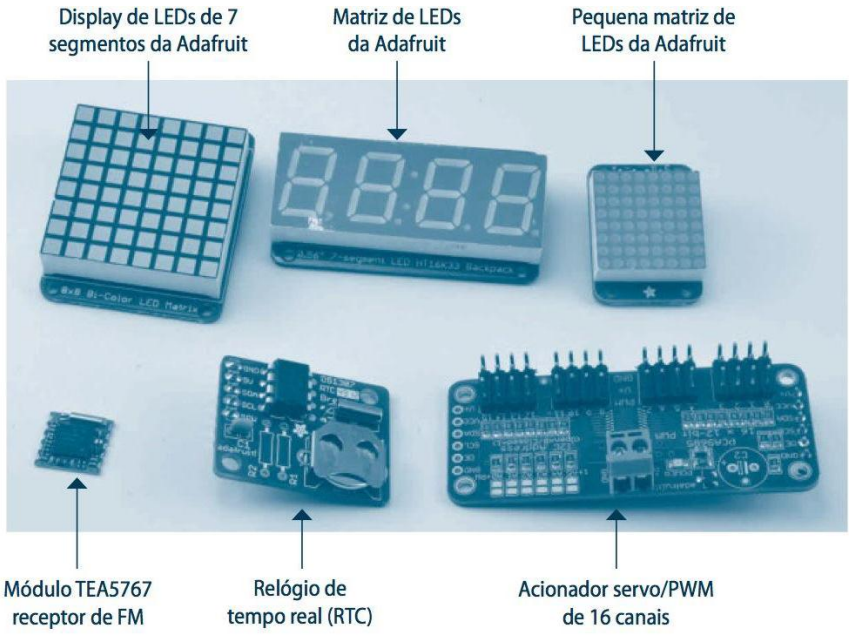

Figura 12 – Alguns dispositivos que possuem comunicação I2C..

Fonte: Monk (2015).

O barramento I2C possui duas vias de comunicação: SDA (*Serial Data*) e SCL (*Serial Clock*) que requerem o uso de resistores de pull-up. A taxa de transferência mínima desta comunicação é de 100kbit/s e a taxa máxima é de 400kbit/s (MONK,2015). A figura 13 mostra o diagrama de tempo desse barramento.

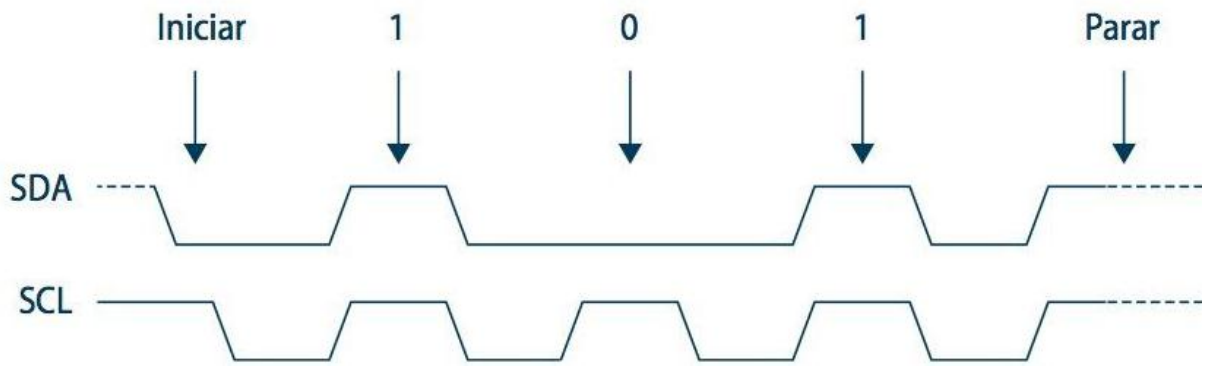

Figura 13 – Diagrama de tempo do I2C.

Fonte: Monk (2015).

A tabela 4 mostra o número correspondente dos pinos SDA e SCL para as diferentes placas de Arduino.

Tabela 4 – Conexões I2C em placa de Arduino.

Placa	Pinos	Notas
Uno	A4 (SDA) e A5 (SCL)	As conexões SCL e SDA próximo de AREF também estão conectadas a A4 e A5.
Leonardo	D2 (SDA) e D3 (SCL)	As conexões SCL e SDA próximo de AREF também estão conectadas a D2 e D3
Mega2560	D20 (SDA) e D21 (SCL)	
Due	D20 (SDA) e D21 (SCL)	O Due também tem um segundo par de conexões I2C, denominadas SDA1 e SCL1.

Fonte: Monk (2015).

2.6 COMUNICAÇÃO 1-WIRE

Esse tipo de comunicação entre um microcontrolador e seu periférico se baseia numa alimentação elétrica chamada parasita que permite ligar dispositivos remotos ao microcontrolador utilizando apenas dois fios; um fio que serve de referência (terra) e outro que combina alimentação elétrica e dados (MONK,2015).

O protocolo de comunicação 1-Wire usa uma linha de dados baseada no conceito mestre e escravo. Na fabricação, cada dispositivo recebe um número de identificação (ID) que representa seu “endereço” de modo que possa ser identificado no barramento. Nesse protocolo de comunicação, o mestre pode chavear a linha de dados entre os modos de entrada e saída para permitir a comunicação nos dois sentidos (MONK,2015).

A figura 14 mostra a conexão elétrica entre o sensor de temperatura DS18B20 e o Arduino usando o conceito de alimentação parasita.

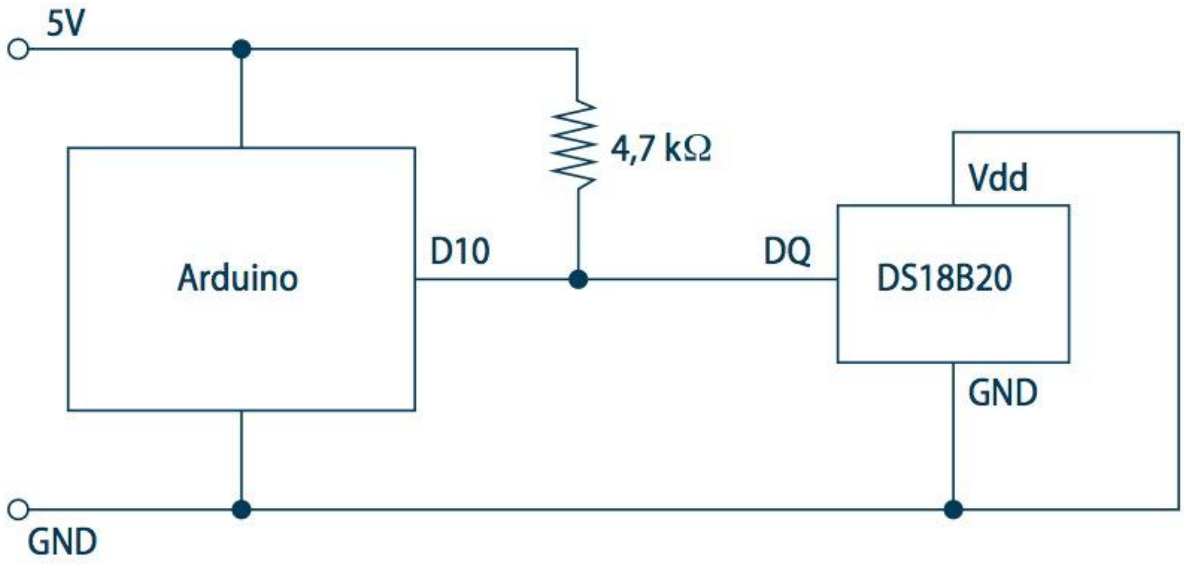

Figura 14 – Conexãodo sensor DS18B20 e o Arduino

Fonte: Monk (2015).

2.7 MÓDULOS PARA ARDUINO UTILIZADOS NESTE TRABALHO

2.7.1 Sensor de Umidade e Temperatura DHT11

O sensor digital DHT11 (Figura 15) mede a temperatura do ambiente entre 0 e 60 ºC através de um termistor interno do tipo NTC e mede a umidade relativa do ar entre 20 a 90% através do sensor de umidade HR202. Este sensor utiliza o protocolo de comunicação 1-Wire. O DHT11 possui de 3 pinos, VCC, DATA e GND. O pino VCC deve ser ligado em 5 VDC e o pino GND deve ser ligado em 0 VDC. O pino digital DATA deve ser ligado a um pino digital do microcontrolador (SUNROM TECHNOLOGIES, 2012). A Tabela 5 apresenta suas especificações técnicas.

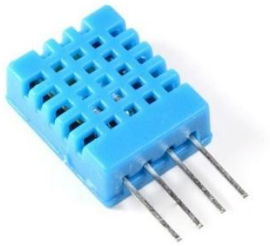

Figura 15 – Sensor DHT11.

Fonte: Sunrom (2012).

Tabela 5 – Características de operação do sensor DHT11.

Modelo	I052115
Chip Principal	DHT11
Faixa de Medição da Umidade Relativa do Ar	20 a 90%
Faixa de Medição de Temperatura do Ar	0 a 60°C
Tensão de Funcionamento	5V DC

Fonte: Autoria própria.

2.7.2 Higrômetro Resistivo

O higrômetro resistivo (Figura 16) é um sensor que mede a quantidade de água presente no solo. Consiste de dois eletrodos que devem ser fixados no solo e um trimpot para ajuste de sensibilidade (limite entre estado seco e úmido).

Uma corrente percorre as duas sondas e a umidade é obtida através da resistência resultante. Quando o solo está seco, a saída do sensor fica em nível lógico alto (5 Volts) e quando o solo está úmido, a saída do sensor fica em nível lógico baixo (0 Volts). A Tabela 6 mostra as condições de operação do higrômetro.

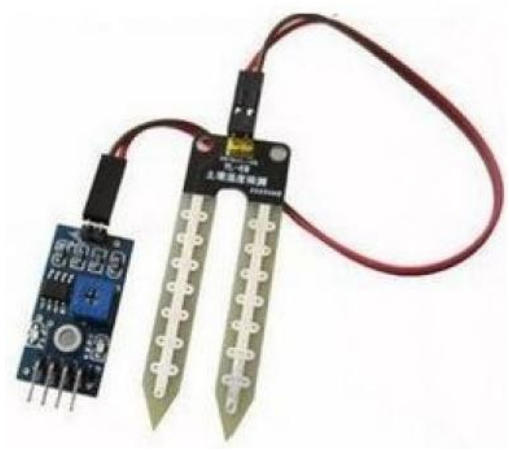

Figura 16– Sensor de umidade do solo.

Fonte: Monk (2012).

Tabela 6 – Características de operação do sensor higrômetro.

Alimentação	3,3 ou 5 V
Corrente Máxima	35mA
Sinal de tensão de saída	0 a 4,2V
Interface	Analógica

Fonte: Autoria própria.

2.7.3 Display LCD

Segundo Zanco (2010) o LCD pode ser classificado como gráfico ou caractere dependendo da forma como é exibido a mensagem no *display*. Os LCDs do tipo caractere geralmente são compatíveis com o código ASCII, ou seja, podem apresentar no *display* mensagens com letras, números e símbolos.

O *display* LCD possui um controlador interno que reconhece o conjunto de instruções transmitidas por quatro ou oito linhas de dados e três linhas de controle (MIYADAIRA,2011).

2.7.3.1 Display LCD 16x2

No mercado existe uma grande variedade de *displays LCD*. Neste trabalho foi utilizado um *display LCD* 16x2 (Figura 17), o que permite exibir duas linhas de texto com até 16 caracteres.

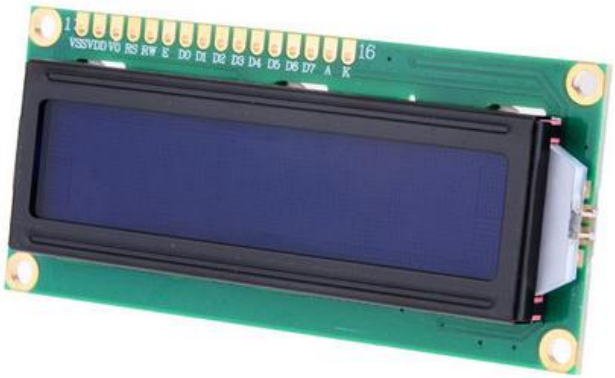

Figura 17 – Display LCD 16x2 .
Fonte: Zanco (2010).

A tensão de alimentação do display pode variar entre 4,7V até 5V. Os pinos 15 e 16 (Tabela 6) são conectados à um LED para iluminação do fundo do display. O pino 3 do LCD é utilizado para fazer o ajuste do contraste e o pino 4 tem a função de

informar se o byte que se encontra na via de dados é um dado ou uma instrução (ZANCO,2010).

Para a leitura ou escrita de dados no *display* é necessário dar um pulso no pino 6, pois ele aplica no pino nível lógico 1 por um tempo curto e depois nível lógico 0. No caso da escrita, o dado deve ficar no barramento por alguns nanossegundos depois do pulso aplicado no pino 6, pois o LCD faz a leitura de dado na borda de descida do pino 6 (ZANCO,2010).

Já no caso de leitura, após ter aplicado nível lógico 1 no pino 6, o dado estará disponível no barramento para ser lido pelo microcontrolador. O pino 5 tem a função de ativar o modo de leitura ou escrita do LCD (ZANCO,2010).

O display LCD possui duas memórias RAMs, sendo elas: DDRAM e CGRAM. Na primeira ficam armazenados dados que serão visualizados no LCD enquanto na outra ficam armazenados caracteres especiais criados pelo usuário (ZANCO,2010).

Tabela 7 – Funções dos pinos do display LCD.

Pino	Nome	Função
1	Vss	Terra
2	Vdd	Positivo (normalmente 5V)
3	Vo	Constrate do LCD. Às vezes também é chamado de Vee
4	RS	Register Select
5	R/W	Read/Write
6	E	Enable
7	D0	Bit 0 do dado a ser escrito no LCD (ou lido dele)
8	D1	Bit 1 do dado a ser escrito no LCD (ou lido dele)
9	D2	Bit 2 do dado a ser escrito no LCD (ou lido dele)
10	D3	Bit 3 do dado a ser escrito no LCD (ou lido dele)
11	D4	Bit 4 do dado a ser escrito no LCD (ou lido dele)
12	D5	Bit 5 do dado a ser escrito no LCD (ou lido dele)
13	D6	Bit 6 do dado a ser escrito no LCD (ou lido dele)
14	D7	Bit 7 do dado a ser escrito no LCD (ou lido dele)
15	A	Anodo do back-light (se existir back-light)
16	K	Catodo do back-light (se existir back-light)

Fonte: Monk (2015).

2.7.4 Módulo Relógio RTC DS1307

O relógio de tempo real DS1307, da sigla RTC (Real Time Clock), é capaz de receber determinada hora e data e mantê-las atualizadas em tempo real**.** A transferência de dados entre o RTC e o Arduino é feita via comunicação I2C. Esse módulo (Figura 18) possui 56 bytes de SRAM, consome pouca energia e vem com uma bateria de lítio (3,0VDC) para evitar perdas de dados caso falte energia (MAXIM INTEGRATED, 2008).

Figura 18 – Módulo relógio.

Fonte: Monk (2012).

2.7.5 SENSOR DE LUMINOSIDADE LDR

O sensor LDR (Resistor Dependente de Luz) ou fotorresitor, libera portadores de carga favorecendo a condução de corrente elétrica quando uma luz incide no sensor (THOMAZINI,2013).

Este sensor é composto pelo sulfeto de cádmio que forma trilhas do material condutor e possui dois terminais que dão acesso para ligação do sensor a um circuito externo. No escuro, a resistência deste sensor é elevada, na ordem de milhões de ohms, mas com a claridade ocorre o oposto, sua resistência diminui consideravelmente (THOMAZINI,2013).

Os LDRs não são polarizados, ou seja, a corrente pode circular nos dois sentidos. Quanto maior for a superfície do LDR maior será a intensidade da corrente circulante e consequentemente maior dissipação de calor (THOMAZINI,2013).

Segundo Thomazini (2013), o tempo de resposta de um fotorresistor é o tempo necessário para que o valor de sua condutância atinja 63% do valor de pico. Quando o LDR recebendo luz, sua resistência atinge um valor de 1000Ω e ao ocorrer interrupção de luz, sua resistência demora cinco segundos para atingir o valor de 1MΩ. Quando houver novamente a iluminação, o valor de resistência de 1MΩ demora dez milissegundos para atingir o valor de 1000Ω novamente. Portanto, este sensor possui um tempo de resposta lenta.

Quanto a dissipação, o fotorresistor (Figura 19), que possui diâmetro de 1cm, dissipa uma potência de 100mW e suporta tensões de até 150V entre seus terminais (THOMAZINI,2013).

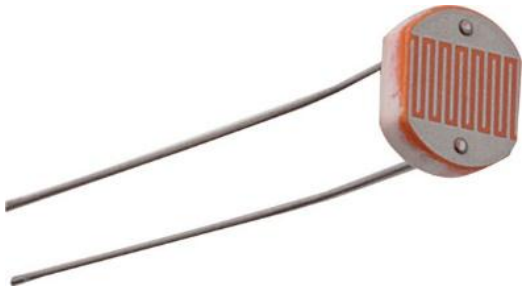

Figura 19 – Sensor de luminosidade LDR.

Fonte: Thomazini (2013).

2.7.6 Cartão de Memória SD

O cartão de memória SD possui uma memória FLASH de tamanho reduzido sendo utilizada em vários tipos de aparelhos eletrônicos para armazenamento permanente de dados (MIYADAIRA,2011).

Os protocolos de comunicação utilizados pelos cartões de memória são o SD e o SPI. O SPI é um protocolo bit a bit e no protocolo SD os dados são transferidos de quatro em quatro bits. O Arduino não suporta o protocolo de comunicação SD, portanto só se comunica com o cartão através do protocolo SPI (MIYADAIRA,2011).

Os cartões SD são classificados com números de 1 a 10, sendo 10 o de maior velocidade. O cartão classe 10 é muito utilizado em filmagens pois vídeos em alta definição operam com velocidade de gravação a partir de 30MB/s. A Figura 206 mostra um cartão SD classe 10.

Figura 20 – Cartão de memória classe 10.

Fonte: Autoria própria

3 MATERIAIS E MÉTODOS

Neste capítulo será abordado o desenvolvimente do *hardware* e a programação do microcontrolador (*software*).

3.1 MONTAGEM

As ligações entre os componentes eletrônicos e o Arduino foram feitas na *protoboard* (matriz de contato). A alimentação do hardware foi feita através do Arduino Mega, o qual está ligado através de porta USB do computador. Mais adiante, será feito um teste para calcular o consumo de energia elétrica da placa junto aos componentes ligados a ela, sendo assim será estimado a bateria necessária para alimentar o circuito de uma forma duradoura.

O Arduino Mega foi utilizado neste projeto pois foram necessários 20 pinos para conectá-lo aos componentes eletrônicos.

A figura 21 mostra dentro de cada flecha a nomenclatura dos pinos de cada componente e em qual pino do Arduino cada um está conectado. Cada fecha mostra a direção do fluxo de dados entre o componente e o Arduino.

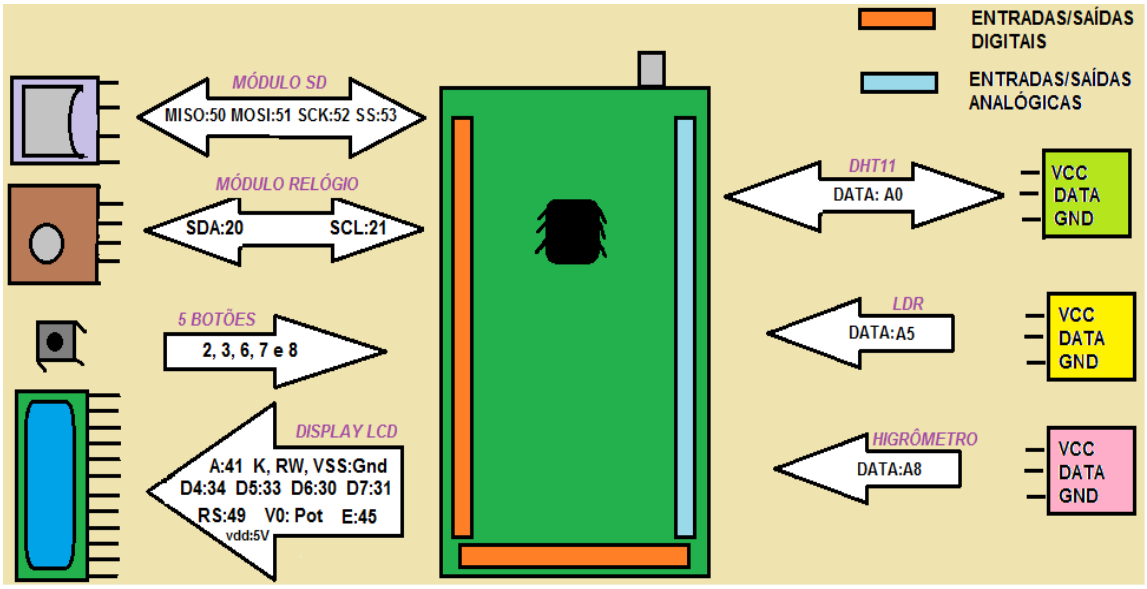

Figura 21- Diagrama do projeto

Fonte: Autoria própria.

3.2 SOFTWARE

O programa foi desenvolvido em linguagem C++ com o compilador IDE para Arduino. O microcontrolador responsável para o funcionamento do *software* é o modelo ATMEGA 2560.

O código fonte foi estruturado utilizando vários laços de rotina, cada uma com sua funcionalidade. No início do código existe um laço com nome “void atualizaMenu(int i)” que é responsável para mostrar o *menu* de opções no *display* LCD. Quando uma das opções é selecionada, o *software* entra no laço de rotina correspondente.

Existe um laço de repetição (*loop*) com nome de “void loop(void)”. Neste laço está a rotina de gravação e configurações de tempo. Se não há interrupção externa (usuário apertar botão “m*enu/select”*), o programa fica o tempo todo nesse laço de repetição fazendo a varredura do tempo. Quando chega a hora de fazer a gravação de dados dos sensores, o programa entra na rotina “grava_SD(t)” responsável por

gravar no cartão SD os valores lidos pelos sensores, com as horas, minutos, segundos, dia, mês e ano daquele exato momento.

O Arduino adormecer, após serem feitas as leituras dos sensores e do RTC, é utilizado o comando “sleep_enable”. Quando for a hora de refazer as leituras, o comando “sleep_disable()” desperta o Arduino.

As bibliotecas utilizadas na programação do Arduino estão representadas pela figura 22.

```
#include "RTClib.h"            // biblioteca do módulo RTC
#include <LiquidCrystal.h>     // biblioteca do display LCD
#include <DS3232RTC.h>         // biblioteca de tempo
#include <TimerOne.h>          // biblioteca para interrupção
#include <Streaming.h>         // biblioteca para streams baseado em caracteres
#include <Time.h>              // biblioteca  de tempo
#include <Wire.h>              // biblioteca para comunicação I2C
#include <SD.h>                // biblioteca para cartão SD
#include <SPI.h>               // biblioteca para comunicação SPI
#include "DHT.h"               // biblioteca do sensor DHT11
```

Figura 22– Bibliotecas do programa.

Fonte: Autoria própria

Ao ligar o datalogger, o display permanece apagado. Ao pressionar o botão “m*enu/select*”, a luz de fundo (conhecida como blacklight) acenderá e uma lista com quatro opções aparece da seguinte maneira:

- 1) AJUSTAR DATA
- 2) AJUSTAR HORA

- 3) TEMP AQUISIÇÃO
- 4) VOLTAR

A figura 23 mostra o fluxograma do menu de opções e a figura 24 ilustra o arranjo dos botões.

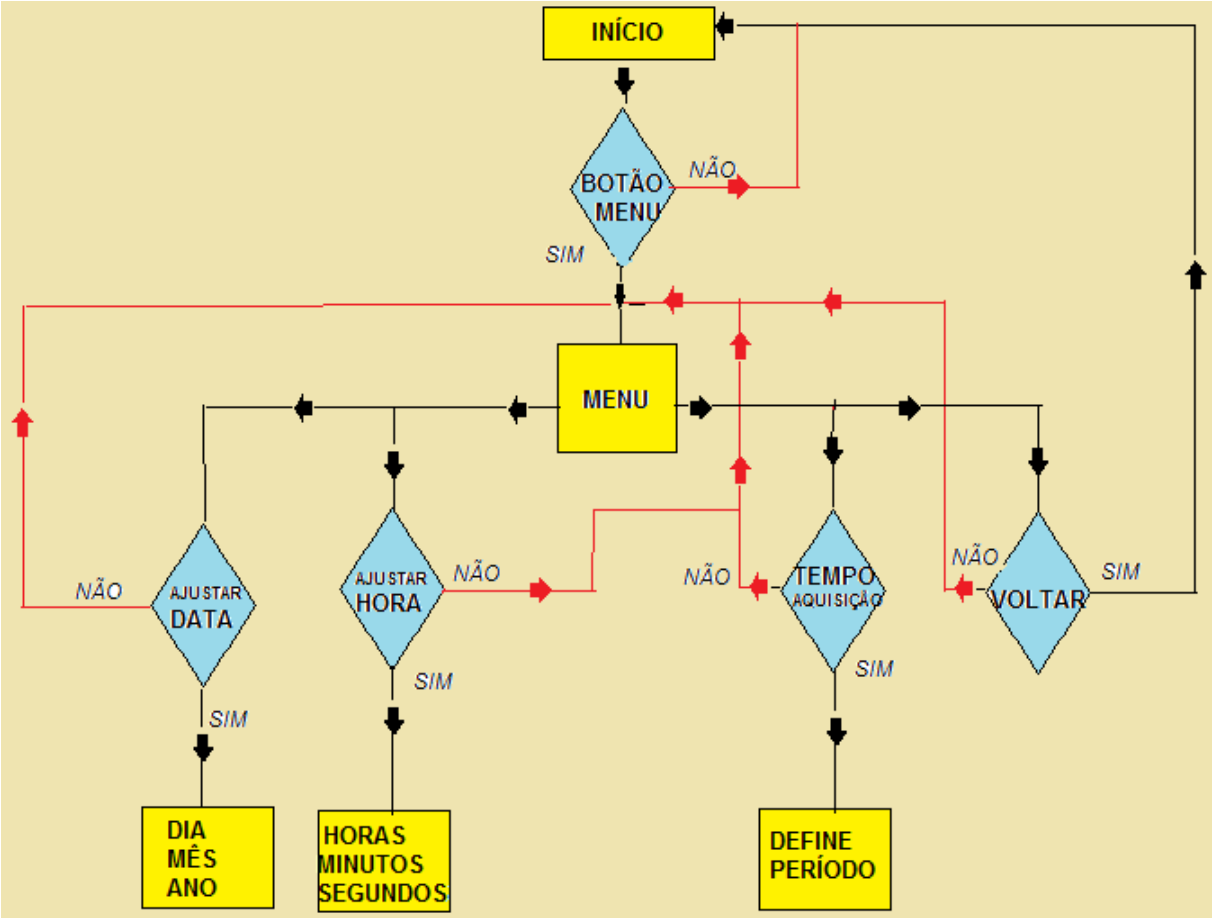

Figura 23– Fluxograma do menu de opções.

Fonte: Autoria própria

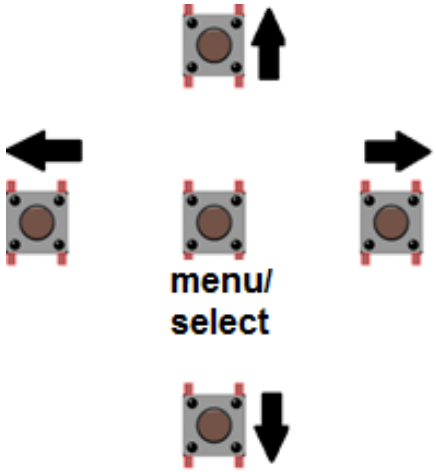

Figura 24– Arranjo dos botões.

Fonte: Autoria própria

O display mostra cada opção da lista por vez. Para selecionar uma delas, basta clicar no botão “m*enu/select”*. O botão “seta para baixo”, quando pressionado, faz aparecer no display a próxima opção da lista. O botão “seta para cima” volta na opção anterior da lista.

Quando a opção “AJUSTAR DATA” é escolhida, será possível configurar o dia, mês e ano do datalogger. A princípio, o dia aparece como primeira opção de configuração. O botão “seta direita” vai mudando as opções dia, mês e ano ordenadamente para configuração, respectivamente. Os botões “seta para cima” e “seta para baixo” neste caso, servirão para incrementar e decrementar valores.

Na opção “AJUSTAR HORA” serão configurados as horas minutos e segundos. Primeiro ajusta-se as horas. Depois com o botão “seta direita” seleciona os minutos e por últimos os segundos.

A terceira opção do menu principal é “TEMP LEITURA”, na qual será ajustado o tempo de aquisição de dados dos sensores. Esse tempo significa o período que os sensores funcionam escolhido pelo operador, ou seja, os sensores farão a leitura de dados e depois o Arduino fica em modo de descanso (modo *sleep*).

Quando a última opção “VOLTAR” é selecionada, o programa sai do *menu* principal e apaga o display, funcionando com as configurações feitas pelo usuário.

No começo do fluxograma da figura 26 os sensores estão em modo de baixo consumo (modo “*sleep*”). Existe um temporizador que faz a contagem de tempo. O

programa verifica se é o momento dos sensores fazerem suas respectivas leituras de dados. Caso não seja o momento, o Arduino continua em modo “*sleep*”. Se for o momento, o Arduino desperta, os sensores fazem as leituras, o programa grava os dados no cartão SD e o Arduino entrra em estado de dormência novamente. A tabela 7 mostra alguns trechos do programa com sua respectiva função.

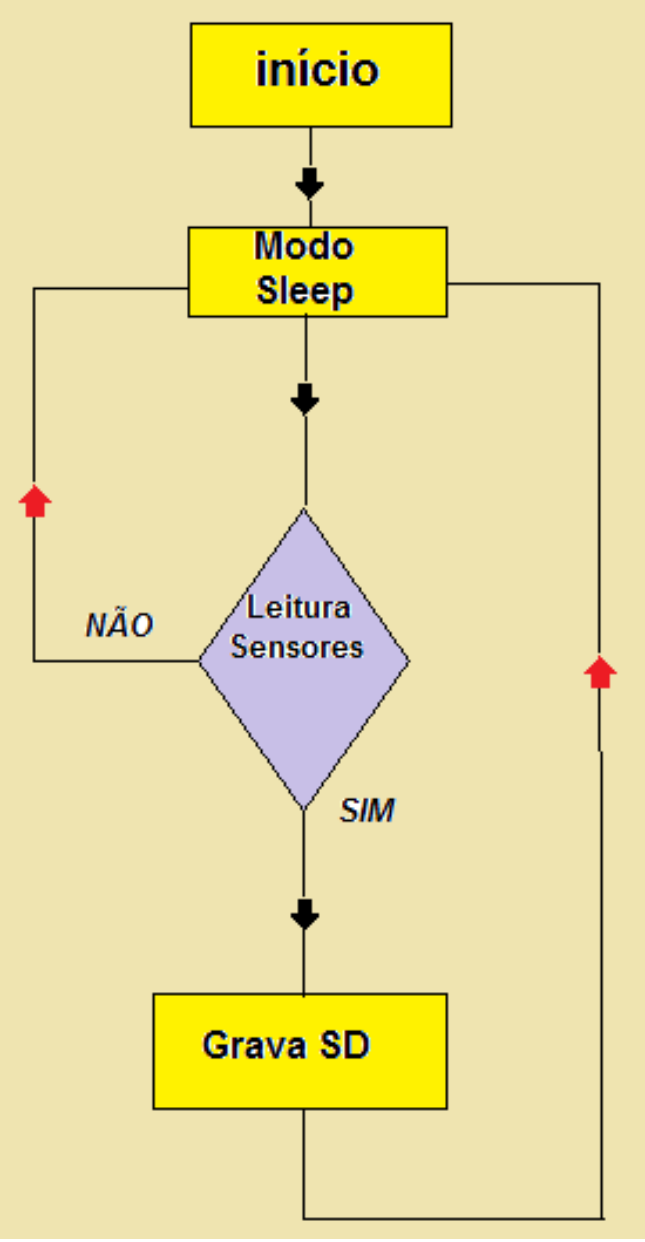

Figura 25– Fluxograma do momento de dormência até a gravação.

Fonte: Autoria própria

Tabela 7- Comandos do software com suas respectivas funções

COMANDO	FUNÇÃO
sleep_enable();	Ativa modo dormir
sleep_disable();	Desativa modo dormir
int sensorValue = analogRead(A0);	Leitura do sensor DHT11
VAL = analogRead(LDR);	Leitura do sensor LDR
soil = map(sensorValue, 485, 1023,100, 0);	Leitura do higrômetro
void grava_SD(time_t t)	Rotina para gravação no cartão de memória
fecha_arquivo();	Término da gravação dos arquivos no cartão

Fonte: Autoria própria.

3.3 CÁLCULO DE CONSUMO

Nesta etapa será estimado o consumo total de energia do circuito. Como o Arduino é responsável por gerenciar todo o sistema, ele será a referência para medição, pois é nele que toda a corrente do circuito passa. O instrumento de medição é um multímetro digital graduado na escala mA (mili amperes) em corrente contínua.

Uma extremidade do *jumper* é conectada na entrada 5V do Arduino, a outra extremidade é ligada na ponta do cabo do multímetro referente a corrente. A ponta do outro cabo do multímetro (terra) é ligada em uma extremidade de um novo *jumper* e a outra extremidade desse *jumper* na entrada da *protoboard* que alimenta todo o circuito eletrônico. Assim tem-se o multímetro em série com o circuito todo. A figura 26 ilustra um multímetro em série com o circuito em funcionamento.

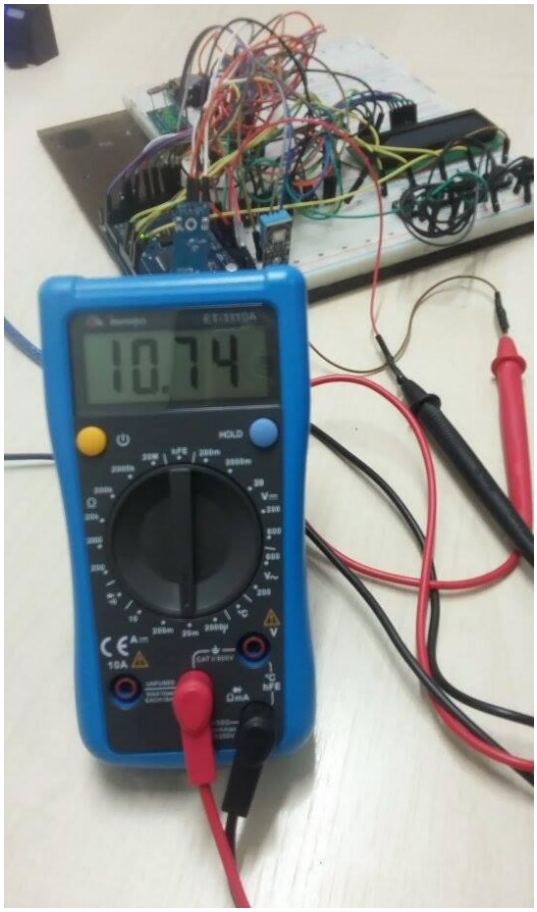

Figura 26– Medição da corrente do circuito.

Fonte: Autoria própria.

Para a medição da corrente, nenhum botão foi apertado, mantendo o circuito funcionando sem interrupções externas.

Como pode ser visto na figura 26, o valor da corrente medida do circuito é de 10,74mA. Sendo assim, podemos fazer o seguinte cálculo para estimar a bateria necessária para o circuito:

1 dia = 24 horas 1Mês = 24h x 30 dias = 720 horas

Se o circuito gasta 10,74 mA, então 720 x 10,74 = 7.732,8 mAh

A bateria (ou pilhas) deve ser de 7.740 mAh para o circuito funcionar durante trinta dias ininterruptamente.

4 RESULTADOS

Após a montagem do projeto e desenvolvida toda a programação, foram feitos dois testes em ambiente fechado durante uma hora cada um. O período de aquisição de dados escolhido para os sensores em cada teste foi de uma hora.

Cada ponto no gráfico refere-se ao período de uma hora, ou seja, o datalogger fez a leitura do sensor e ficou em modo *sleep* durante 59 minutos para fazer a leitura novamente. O tempo total de teste foi de 5 horas, começando ás 10:00 horas e terminando ás 15:00 horas.

As 11:00 horas a terra em que se encontra o sensor higrômetro, será molhada.

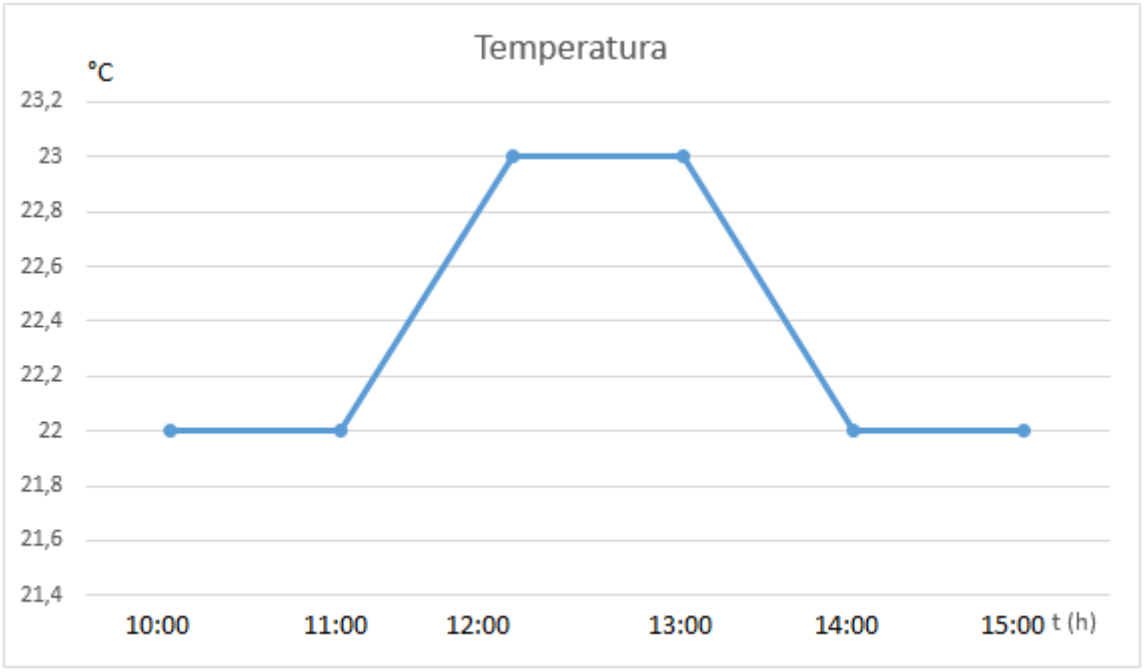

Figura 27– Temperatura do ambiente

Fonte: Autoria própria.

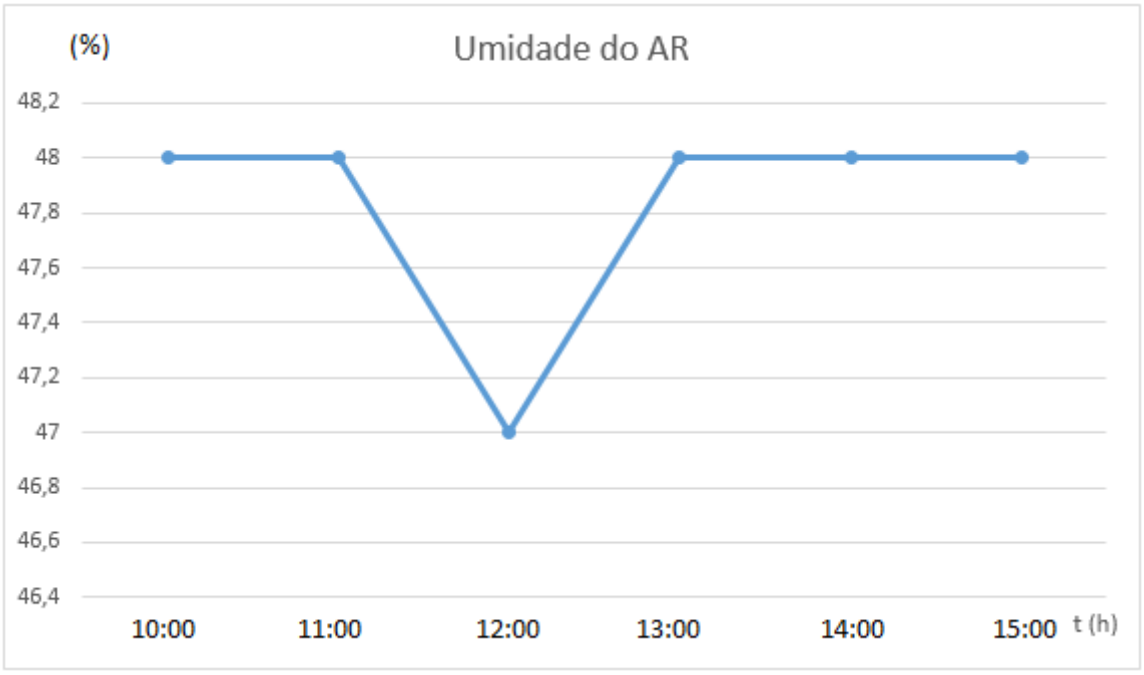

Figura 28– Umidade relativa do ar.

Fonte: Autoria própria.

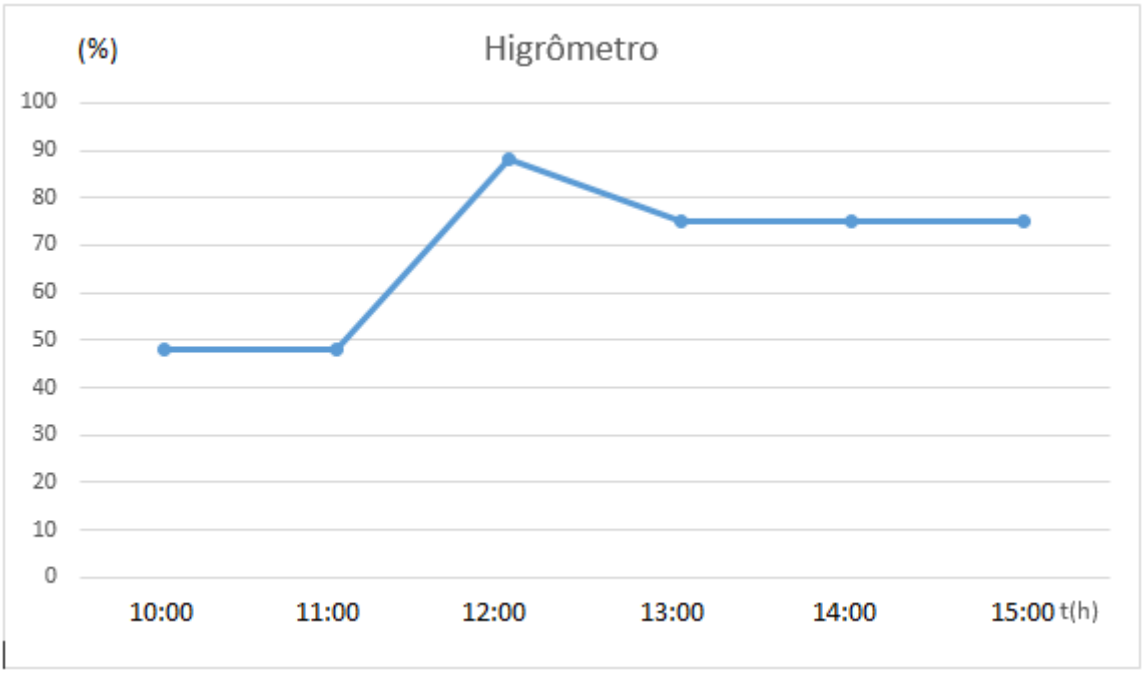

Figura 29– Umidade relativa do solo.

Fonte: Autoria própria.

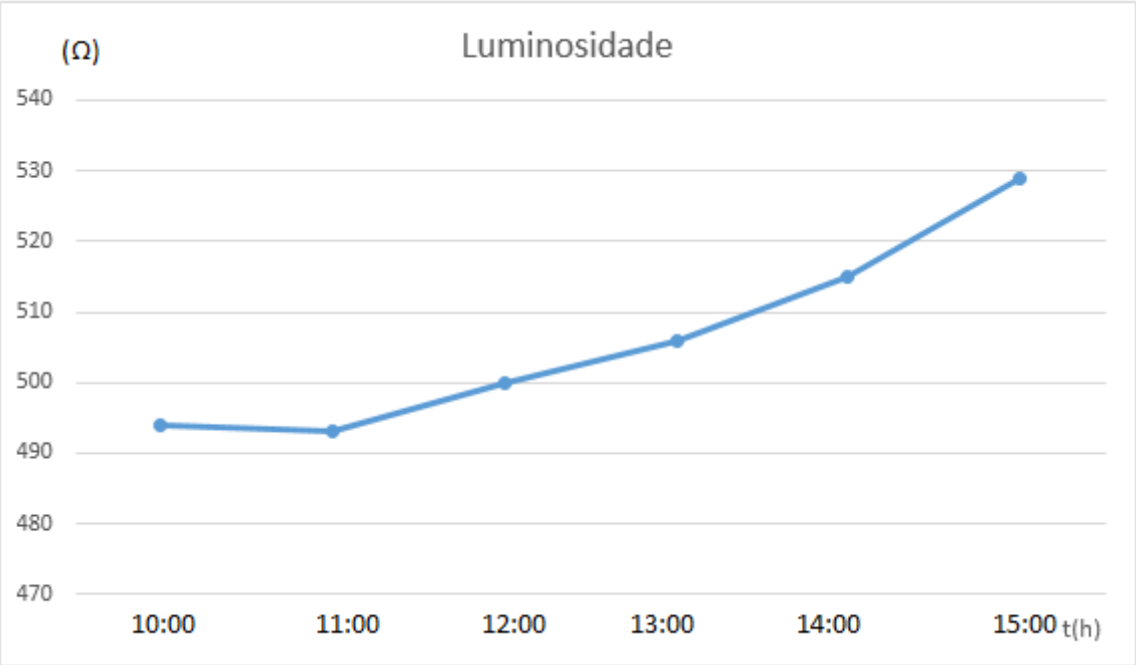

Figura 30- Variação da resistência devido a intensidade luminosa.

Fonte: Autoria própria.

5 CONCLUSÃO

Neste trabalho foram estudados vários tipos de sensores e o Arduino. Foi montado um datalogger para as seguintes aquisições de dados metereológicos: Temperatura do ambiente, umidade relativa do ar, intensidade de luminosidade e umidade relativa do solo.

Com os gráficos obtidos, é possível analisar as variações dos sensores com as mudanças climáticas. Se as condições do ambiente se mantém constante os gráficos se comportam de forma linear. Conforme as condições variam, os sensores respondem com oscilações no gráfico.

No gráfico da temperatura, nota-se o aumento de um grau no horário do meio dia. Para a umidade relativa do ar houve pouca variação, apenas de 1% de umidade, sendo praticamente estável.

O gráfico do higrômetro mostrou grande variação no horário das 11:00 horas, visto que a terra foi molhada. O gráfico do sensor ldr foi aumentando gradativamente tendo seu máximo no horário das 15:00 horas.

REFERÊNCIAS

MCROBERTS, Michael. **Arduino básico**. 1.ed. São Paulo: Novatec,2011.

MIYADAIRA, Alberto Noboru. **Microcontroladores PIC18**. 2.ed. São Paulo: Érica,2011.

MIXEQUIPAMENTOS. **Datalogger DL2e**. DataloggerDisponível em: < www.mixequipamentos.com> Acesso em: 26 julho 2015.

MONTEIRO, José Eduardo B.A. **Agrometeorologia dos cultivos**. O fator metereológico na produção agrícola. 1ed. Brasília: Inmet,2009.

MONK, Simon. **Programação com Arduino: começando com sketches-série tekne**. 1ed.São Paulo: Grupo A, 2013.

MONK, Simon. **Programação com Arduino 2: começando com sketches-série tekne**. 1ed.São Paulo: Grupo A, 2015.

NOVUS**. Datalogger LogBox- RHT-LCD**. Disponível em: < http://www.novus.com.br/site/default.asp>. Acesso em: 27 setembro 2015.

SENTELHAS, P. C.; MONTEIRO, J. E. B. A. **Agrometeorologia dos cultivos- O fator meteorológico na produção agrícola**. INMET, 2009.

THOMAZINI, Daniel. **Sensores industriais**. **Fundamentos e aplicações.** 4ed. São Paulo: Editora Èrica,2013.

APÊNDICE

```
#define isBissexto(a) (a%4==0 || a%100!=0 || a%400==0)

   #define LED_DEBUG 5
   #define BT_SELECT 2
   #define BT_TOP    3
   #define BT_LEFT   6
   #define BT_BOTTOM 7
   #define BT_RIGHT  8
    #define apaga 41
    #include "RTClib.h"

   #include <LiquidCrystal.h>

 LiquidCrystal lcd(49, 45, 35, 33, 28, 29);

   RTC_DS1307 rtc;

   unsigned long cadaSegundo = 0L;
   unsigned long piscaAjuste = 0L;

   DateTime despertador;
   boolean despertadorLigado = false;

#include <avr/interrupt.h>
#include <avr/sleep.h>
#include <DS3232RTC.h>
#include <Streaming.h>
#include <Time.h>
#include <Wire.h>
#include <SD.h>
#include <SPI.h>
#include "DHT.h"
#define DHTPIN A0
#define DHTTYPE DHT11
DHT dht(DHTPIN, DHTTYPE);
```

```
int soil=0;
 int LDR = A7;
int VAL = A7;
int AL = 0;
int BA = 0;
int CS_PIN = 4;
 byte mode;
File file;
 void setup(void)
{
 set_sleep_mode(mode);
  TCCR1A = 0;
  TCCR1B = 0;
  TCCR1B |= (1<<CS10)|(1 << CS12);
TCNT1 = 0xC2F7;
 TIMSK1 |= (1 << TOIE1);
    dht.begin();
  pinMode(LED_DEBUG, OUTPUT);
      pinMode(apaga, OUTPUT);
      pinMode(BT_SELECT, INPUT);
      pinMode(BT_TOP, INPUT);
      pinMode(BT_LEFT, INPUT);
      pinMode(BT_BOTTOM, INPUT);
      pinMode(BT_RIGHT, INPUT);
     digitalWrite(apaga, LOW);
      digitalWrite(BT_SELECT, HIGH);
      digitalWrite(BT_TOP, HIGH);
      digitalWrite(BT_LEFT, HIGH);
      digitalWrite(BT_BOTTOM, HIGH);
      digitalWrite(BT_RIGHT, HIGH);

      lcd.begin(16, 2);
      lcd.print("RELOGIO DIGITAL");
      lcd.setCursor(0, 1);
      lcd.print("INICIANDO...");
      delay(100);

    #ifdef AVR
```

```
      Wire.begin();
    #else
      Wire1.begin();
    #endif
      rtc.begin();
if (! rtc.isrunning()) {
      rtc.adjust(DateTime(F(__DATE__), F(__TIME__)));
       }
    despertador = DateTime(0,0,0,0,0,0);
        despertadorLigado = false;
     cadaSegundo = millis() + 1000L;
   //Inicializa o cartao SD
  inicia_SD();
 setSyncProvider(RTC.get);
  Serial << F("Sincronizando com o RTC...");
  if (timeStatus() != timeSet) Serial << F(" Falha!");
  Serial << endl;
}
void loop(void)
{
  VAL = analogRead(LDR);
   int sensorValue = analogRead(A8);
     sensorValue = constrain(sensorValue, 485, 1023);
      soil = map(sensorValue, 485, 1023, 100, 0);
      Serial.print("Umidade do solo em: ");
       Serial.print(soil);
  Serial.println("%");
  float u = dht.readHumidity();
  float e = dht.readTemperature();
  Serial.print("Umidade: ");
    Serial.print(u);
    Serial.print(" %t");
    Serial.print("Temperatura: ");
    Serial.print(e);
    Serial.println(" *C");

    if (VAL<512)
 {
```

```
  BA=VAL; //Escreva no serial Monitor
 }
 else if(VAL > 700)
{
AL=VAL;
}
  static time_t tLast;
  time_t t;
  tmElements_t tm;
if (Serial.available() >= 12) {
    int y = Serial.parseInt();
    if (y >= 100 && y < 1000)
      Serial<<F("Erro: Ano deve ter dois ou quatro digitos!") <<endl;
    else {
      if (y >= 1000)
        tm.Year = CalendarYrToTm(y);
      else    //(y < 100)
        tm.Year = y2kYearToTm(y);
      tm.Month = Serial.parseInt();
      tm.Day = Serial.parseInt();
      tm.Hour = Serial.parseInt();
      tm.Minute = Serial.parseInt();
      tm.Second = Serial.parseInt();
      t = makeTime(tm);
      RTC.set(t);
      setTime(t);
      Serial << F("Horario modificado para: ");
      printDateTime(t);
      Serial << endl;
      while (Serial.available() > 0) Serial.read();
    }
  }

  t = now();
  if (t != tLast) {
    tLast = t;
    printDateTime(t);
```

```
    if (second(t) == 0)
    {
      int sensorValue = analogRead(A0);
       float u = dht.readHumidity();
      float e = dht.readTemperature();

      Serial << F("  ") << e << F(" C  ") << u << F(" F");
      Serial.println("\nGravando dados no cartao SD...");
      grava_SD(t);
    }
    Serial << endl;
  }
   // Verificando se botao select foi pressionado
      if (digitalRead(BT_SELECT))
       {
        digitalWrite(apaga, LOW);
      if (cadaSegundo < millis())
       {
         lcdMostraRelogio();
         cadaSegundo = millis() + 1000L;
       }
      verificaDespertador();
      }
      else
      {
        digitalWrite(apaga, HIGH);
      TCNT1 = 0xC2F7;
        delay(300);

        // interrompe funcionamento e entra no menu
        lcdMostraMenu();
      }
}
  void lcdClear()
    {
      for (unsigned j = 0; j < 2; j++)
      {
        lcd.setCursor(0, j);
```

```
      for (unsigned i = 0; i < 16; i++) lcd.print(" ");
    }
  }

  void lcdMostraRelogio()
  {
    DateTime now = rtc.now();
    lcdClear();
    lcd.setCursor(0,0);
    lcd.print("DATA: ");
    if (now.day() < 10) lcd.print("0");
    lcd.print(now.day());
    lcd.print("/");
    if (now.month() < 10) lcd.print("0");
    lcd.print(now.month());
    lcd.print("/");
    lcd.print(now.year());
    lcd.print(" ");
   lcd.setCursor(0,1);
    lcd.print("HORA: ");
    if (now.hour() < 10) lcd.print("0");
    lcd.print(now.hour());
    lcd.print(":");
    if (now.minute() < 10) lcd.print("0");
    lcd.print(now.minute());
    lcd.print(":");
    if (now.second() < 10) lcd.print("0");
    lcd.print(now.second());
    lcd.print(" ");
    lcd.print(despertadorLigado ? "D" : " ");
  }
void lcdMostraMenu()
  {
    boolean voltar = false;
    int idx = 0;
    atualizaMenu(idx);
    while(!voltar)
    {
```

```
    if (!digitalRead(BT_TOP) && idx > 0) { idx--; atualizaMenu(idx); }
    else if (!digitalRead(BT_BOTTOM) && idx < 3) { idx++; atualizaMenu(idx); }
    else if (!digitalRead(BT_SELECT))
    {
      if (idx == 0) rtcAjustaData();
      else if (idx == 1) rtcAjustaHora();
      else if (idx == 2) rtcAjustaAlarme();
      else if (idx == 3) voltar = true;

      atualizaMenu(idx);
    }
  }
  lcdClear();
  lcd.setCursor(0,0);
  lcd.print("VOLTANDO...");
  delay(1000);
}

void atualizaMenu(int i)
{
  lcdClear();
  lcd.setCursor(0,0);
  lcd.print("MENU PRINCIPAL");
  lcd.setCursor(0,1);
  if (i==0)
    lcd.print("1) AJUSTAR DATA");
  else if (i ==1)
    lcd.print("2) AJUSTAR HORA");
  else if (i ==2)
    lcd.print("3) TEMP LEITURA");
  else if (i ==3)
    lcd.print("4) VOLTAR");
  delay(300);
}

void atualizaMenuData(int d, int m, int y, int i)
{
  DateTime now = rtc.now();
```

```
  lcdClear();
  lcd.setCursor(0,0);
  lcd.print("AJUSTAR DATA");
  lcd.setCursor(0,1);
  if (i == 0 && piscaAjuste < millis())
  {
    lcd.print("  ");
    piscaAjuste = millis() + 500L;
  }
  else
  {
    if (d < 10) lcd.print("0");
    lcd.print(d);
  }
  lcd.print("/");
  if (i == 1 && piscaAjuste < millis())
  {
    lcd.print("  ");
    piscaAjuste = millis() + 500L;
  }
  else
  {
    if (m < 10) lcd.print("0");
    lcd.print(m);
  }
  lcd.print("/");
  if (i == 2 && piscaAjuste < millis())
  {
    lcd.print("    ");
    piscaAjuste = millis() + 500;
  }
  else
  {
    lcd.print(y);
  }
  lcd.print(" ");
  delay(300);
}
```

```
void rtcAjustaData()
{
  boolean voltar = false;

  unsigned long passou = millis() + 100;
  DateTime now = rtc.now();

  int limite = 31;
  int pIdx[3];
  int idx = 0;


  pIdx[0] = now.day();
  pIdx[1] = now.month();
  pIdx[2] = now.year();


  delay(300);
  while(!voltar)
  {
    if (passou < millis())
    {
      atualizaMenuData(pIdx[0], pIdx[1], pIdx[2], idx);
      passou = millis() + 100;
    }

    if (idx == 0)
    {
      // Para o dia tem que verificar o mes e se o ano é bissexto
      if ((pIdx[1] == 1 || pIdx[1] == 3 || pIdx[1] == 5
          || pIdx[1] == 7 || pIdx[1] == 8 || pIdx[1] == 10 || pIdx[1] == 12)) limite = 31;
      else if ((pIdx[1] == 4 || pIdx[1] == 6 || pIdx[1] == 9
          || pIdx[1] == 11)) limite = 30;
      else if (pIdx[1] == 2 && !isBissexto(pIdx[2])) limite = 28;
      else if (pIdx[1] == 2 && isBissexto(pIdx[2])) limite = 29;
      else limite = 1;
```

```
    }
    else if (idx == 1) limite = 12;
    else if (idx == 2) limite = 9999;
   if (!digitalRead(BT_BOTTOM) && pIdx[idx] > 0)
    {
      pIdx[idx]--;
      atualizaMenuData(pIdx[0], pIdx[1], pIdx[2], idx);
      //delay(100);
    }
    else if (!digitalRead(BT_TOP) && pIdx[idx] < limite)
    {
      pIdx[idx]++;
      atualizaMenuData(pIdx[0], pIdx[1], pIdx[2], idx);
      //delay(100);
    }
    else if (!digitalRead(BT_LEFT) && idx > 0)
    {
      idx--;
      atualizaMenuData(pIdx[0], pIdx[1], pIdx[2], idx);
    }
    else if (!digitalRead(BT_RIGHT) && idx < 2)
    {
      idx++;
      atualizaMenuData(pIdx[0], pIdx[1], pIdx[2], idx);
    }
    else if (!digitalRead(BT_SELECT))
    {
      now = rtc.now();
      rtc.adjust(DateTime(pIdx[2], pIdx[1], pIdx[0], now.hour(), now.minute(), now.second()));
      delay(100);
      voltar = true;
    }
  }
  lcdClear();
  lcd.setCursor(0,0);
  lcd.print("SALVANDO DATA...");
  delay(1000);
}
```

```
void atualizaMenuHora(int h, int m, int s, int i)
{
  DateTime now = rtc.now();
  lcdClear();
  lcd.setCursor(0,0);
  lcd.print("AJUSTAR HORA");
  lcd.setCursor(0,1);
  if (i == 0 && piscaAjuste < millis())
  {
    lcd.print("  ");
    piscaAjuste = millis() + 500L;
  }
  else
  {
    if (h < 10) lcd.print("0");
    lcd.print(h);
  }
  lcd.print(":");
  if (i == 1 && piscaAjuste < millis())
  {
    lcd.print("  ");
    piscaAjuste = millis() + 500L;
  }
  else
  {
    if (m < 10) lcd.print("0");
    lcd.print(m);
  }
  lcd.print(":");
  if (i == 2 && piscaAjuste < millis())
  {
    lcd.print("    ");
    piscaAjuste = millis() + 500;
  }
  else
  {
    if (s < 10) lcd.print("0");
```

```
    lcd.print(s);
  }
  lcd.print(" ");
  delay(300);
}

void rtcAjustaHora()
{
  boolean voltar = false;

  unsigned long passou = millis() + 100;
  DateTime now = rtc.now();

  int limite = 31;
  int pIdx[3];
  int idx = 0;

  pIdx[0] = now.hour();
  pIdx[1] = now.minute();
  pIdx[2] = now.second();

  delay(300);
  while(!voltar)
  {
    if (passou < millis())
    {
      atualizaMenuHora(pIdx[0], pIdx[1], pIdx[2], idx);
      passou = millis() + 100;
    }

    if (idx == 0) limite = 24;
    else limite = 59;

    if (!digitalRead(BT_BOTTOM) && pIdx[idx] > 0)
    {
      pIdx[idx]--;
      atualizaMenuHora(pIdx[0], pIdx[1], pIdx[2], idx);
    }
```

```
    else if (!digitalRead(BT_TOP) && pIdx[idx] < limite)
    {
      pIdx[idx]++;
      atualizaMenuHora(pIdx[0], pIdx[1], pIdx[2], idx);
    }
    else if (!digitalRead(BT_LEFT) && idx > 0)
    {
      idx--;
      atualizaMenuHora(pIdx[0], pIdx[1], pIdx[2], idx);
    }
    else if (!digitalRead(BT_RIGHT) && idx < 2)
    {
      idx++;
      atualizaMenuHora(pIdx[0], pIdx[1], pIdx[2], idx);
    }
    else if (!digitalRead(BT_SELECT))
    {
      now = rtc.now();
      rtc.adjust(DateTime(now.year(), now.month(), now.day(), pIdx[0], pIdx[1], pIdx[2]));
      delay(100);
      voltar = true;
    }
  }
  lcdClear();
  lcd.setCursor(0,0);
  lcd.print("SALVANDO HORA...");
  delay(1000);
}

void atualizaMenuAlarme(int i)
{
  lcdClear();
  lcd.setCursor(0,0);
  lcd.print("LEITURA SENSORES");
  lcd.setCursor(0,1);
  if (i==0)
    lcd.print("1) AJUSTAR");
  else if (i ==1)
```

```
  {
    lcd.print("2) ");
    if (despertadorLigado) lcd.print("DESLIGAR");
    else lcd.print("LIGAR");
  }
  else if (i ==2)
    lcd.print("3) VOLTAR");
  delay(300);
}

void rtcAjustaAlarme()
{
  boolean voltar = false;
  int idx = 0;
  atualizaMenuAlarme(idx);
  while(!voltar)
  {
    if (!digitalRead(BT_TOP) && idx > 0) { idx--; atualizaMenuAlarme(idx); }
    else if (!digitalRead(BT_BOTTOM) && idx < 2) { idx++; atualizaMenuAlarme(idx); }
    else if (!digitalRead(BT_SELECT))
    {
      if (idx == 0) rtcAjustaDespertador();
      else if (idx == 1) rtcLigaDespertador();
      else if (idx == 2) voltar = true;

      atualizaMenuAlarme(idx);
    }
  }
  lcdClear();
  lcd.setCursor(0,0);
  lcd.print("VOLTANDO...");
  delay(1000);
}

void rtcAjustaDespertador()
{
  boolean voltar = false;
```

```
  unsigned long passou = millis() + 100;
  DateTime now = rtc.now();
 int limite = 31;
  int pIdx[3];
  int idx = 0;
pIdx[0] = now.hour();
  pIdx[1] = now.minute();
  pIdx[2] = now.second();

  delay(300);
  while(!voltar)
  {
    if (passou < millis())
    {
      atualizaMenuHora(pIdx[0], pIdx[1], pIdx[2], idx);
      passou = millis() + 100;
    }
  if (idx == 0) limite = 24;
    else limite = 59;

    if (!digitalRead(BT_BOTTOM) && pIdx[idx] > 0)
    {
      pIdx[idx]--;
      atualizaMenuHora(pIdx[0], pIdx[1], pIdx[2], idx);
    }
    else if (!digitalRead(BT_TOP) && pIdx[idx] < limite)
    {
      pIdx[idx]++;
      atualizaMenuHora(pIdx[0], pIdx[1], pIdx[2], idx);
    }
    else if (!digitalRead(BT_LEFT) && idx > 0)
    {
      idx--;
      atualizaMenuHora(pIdx[0], pIdx[1], pIdx[2], idx);
    }
    else if (!digitalRead(BT_RIGHT) && idx < 2)
    {
      idx++;
```

```
        atualizaMenuHora(pIdx[0], pIdx[1], pIdx[2], idx);
      }
      else if (!digitalRead(BT_SELECT))
      {
        now = rtc.now();
        despertador = DateTime(now.year(), now.month(), now.day(), pIdx[0], pIdx[1], pIdx[2]);
        despertadorLigado = true;
        delay(100);
        voltar = true;
      }
    }
    lcdClear();
    lcd.setCursor(0,0);
    lcd.print("SALVANDO ...");
    delay(1000);
  }

  void rtcLigaDespertador()
  {
    despertadorLigado = !despertadorLigado;
    delay(300);
  }
void verificaDespertador()
  {
    unsigned long tempoAlarme = millis() + 500L;
    DateTime now = rtc.now();

    if (despertadorLigado)
    {
      if (now.hour() == despertador.hour() && now.minute() == despertador.minute())
      {
      while(digitalRead(BT_SELECT) && digitalRead(BT_TOP) && digitalRead(BT_BOTTOM)
            && digitalRead(BT_LEFT) && digitalRead(BT_RIGHT))
        {

          lcdClear();
            lcd.setCursor(0,0);
            lcd.print("");
```

```
            if (tempoAlarme < millis())
            {
              lcd.setCursor(0,4);
              lcd.print("");
              if (tempoAlarme + 250 < millis()) tempoAlarme = millis() + 500L;
            }
            else
            {
             }
            delay(100);
          }
          despertadorLigado = false;
          delay(500);
        }
      }
    }

void printDateTime(time_t t)
{
  printI00(day(t), 0);
  Serial << monthShortStr(month(t)) << _DEC(year(t));
  Serial << ' ';
  //printTime(t);
  printI00(hour(t), ':');
  printI00(minute(t), ':');
  printI00(second(t), ' ');
}

//Grava dados no cartao SD
void grava_SD(time_t t)
{
  VAL = analogRead(LDR);
  int sensorValue = analogRead(A0);
   float u = dht.readHumidity();
  float e = dht.readTemperature();
  abre_arquivo_gravacao("data.txt");
  file.print("Data: ");
```

```
file.print(day(t));
file.print("/");
if (month(t) < 10)
{
  file.print("0");
}
file.print(month(t));
file.print("/");
file.print(year(t));
file.print(" Hora: ");
if (hour(t) < 10)
{
  file.print("0");
}
file.print(hour(t));
file.print(":");
if (minute(t) < 10)
{
  file.print("0");
}
file.print(minute(t));
file.print(":");
if (second(t) < 10)
{
  file.print("0");
}
file.println(second(t));
 file.println();
  file.println("_____");
file.print(" Umidade do SOLO em: ");
 file.print(soil);
file.println("%");
file.println();
file.print(" A temperatura do ambiente está em: ");
file.print(e);
file.println(" *C");
file.println();
 file.print(" Umidade do AR em: ");
```

```
  file.print(u);
   file.println("%");
   file.println();
    file.println(" A luminosidade do ambiente está: ");
file.println();
 file.print(" Alta ");
 file.println(AL);
 file.print(" Baixa ");
 file.println(BA);
file.println();
  fecha_arquivo();
  sleep_enable(); // ativa o recurso para dormir
    sei();
    sleep_cpu();
    sleep_disable();
}
//Correcao para imprimir "00" ao inves de "0" caso
//o valor seja menor do que 10
void printI00(int val, char delim)
{
  if (val < 10) Serial << '0';
  Serial << _DEC(val);
  if (delim > 0) Serial << delim;
  return;
}
 void inicia_SD()
{
  pinMode(CS_PIN, OUTPUT);

  if (SD.begin())
  {
  } else
  {
    return;
  }
}
 int abre_arquivo_gravacao(char filename[])
{
```

```
  file = SD.open(filename, FILE_WRITE);

  if (file)
  {
    return 1;
  } else
  {
    return 0;
  }
}

void fecha_arquivo()
{
  if (file)
  {
    file.close();
  }
  delay (2000);
}
```