MINISTÉRIO DA CIÊNCIA E TECNOLOGIA
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

APRIMORAMENTO DE PLATAFORMA DE COLETA DE DADOS AMBIENTAIS DE BAIXO CUSTO – MODELO DEDICADO COM MODOS DE TRANSMISSÃO DE DADOS E DE ARMAZENAMENTO

Andrew Medeiros de Campos

Relatório final de iniciação científica do programa PIBIC, orientado pelo Dr. Ricardo Toshiyuki Irita e coorientado pelo Dr. Waldeir Amaral Vilela.

INPE

São José dos Campos

2021

MINISTÉRIO DA CIÊNCIA E TECNOLOGIA
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

APRIMORAMENTO DE PLATAFORMA DE COLETA DE DADOS AMBIENTAIS DE BAIXO CUSTO – MODELO DEDICADO COM MODOS DE TRANSMISSÃO DE DADOS E DE ARMAZENAMENTO

Andrew Medeiros de Campos

Relatório final de iniciação científica do programa PIBIC, orientado pelo Dr. Ricardo Toshiyuki Irita e coorientado pelo Dr. Waldeir Amaral Vilela.

INPE

São José dos Campos

2021

RESUMO

Este relatório final é parte de um projeto de Iniciação Científica do programa PIBIC desenvolvido no Instituto Nacional de Pesquisas Espaciais (INPE), onde o principal objetivo foi inserir melhorias e um sistema de coleta de dados de baixo custo em desenvolvimento pelo Grupo de Dispositivos Fotovoltaicos (GDF) do Instituto. A principal melhoria no sistema consistiu na inserção de um dispositivo de transmissão de dados. O monitoramento das grandezas ambientais, tais como umidade e temperatura da atmosfera, radiação solar, chuva e vento são cruciais para qualquer tipo de planejamento da utilização do potencial energético em uma sociedade, auxiliando a delimitar a escolha de melhores ações devido ao impacto ambiental. As mudanças climáticas tem levado os países a buscarem alternativas energéticas às fontes fósseis como a energia fotovoltaica ou eólica, cujo o conhecimento das grandezas ambientais no local da implantação destas usinas são fundamentais no planejamento e desenvolvimentos dos projetos. Atualmente, os principais dados de grandezas ambientais são obtidos por modelos computacionais pouco representativos da realidade local ou coletados por plataformas de alto custo distantes entre si, exigindo a interpolação de dados. O aprimoramento de uma unidade de coleta de dados de baixo custo para uso em uma PCD experimental irá proporcionar a melhora na qualidade dos dados gerados assim como facilitar a aquisição de um volume de dados maior, neste intuito, o laboratório GDF/COPDT (Grupo de Dispositivos Fotovoltaicos) do INPE de São José dos Campos mantém um projeto de desenvolvimento de uma PCD experimental de baixo custo com o objetivo de auxiliar na pesquisa de potencial fotovoltaico. Atualmente os dados obtidos são armazenados na memória do próprio dispositivo de aquisição de dados instalado na PCD e o acesso dos dados é feito no local que é de difícil acesso. Viu-se, então, a necessidade de desenvolver um módulo que facilitasse o acesso aos dados por meio da transmissão sem fio. Deste modo, é apresentado neste trabalho o início do desenvolvimento de um módulo de coleta de dados de baixo custo com transmissão de dados para as PCDs a partir do estudo comparativo de diversas formas de transmissão de dados compatíveis com microprocessadores da família ATMega a fim de se encontrar a tecnologia que melhor se adequasse as PCDs em desenvolvimento e então a implementação da mesma em uma plataforma de testes. Foram comparadas as seguintes tecnologias: módulo HC-06 para transmissão Bluetooth, módulo nRF24L01 para transmissão por modulação GFSK de protocolo próprio além dos módulos ESP-01 e NodeMCU para transmissão Wi-Fi. Essas tecnologias foram analisadas por meio de testes de latência, velocidade e distância máxima de transmissão. E como resultado, a melhor solução obtida foi do módulo ESP-01 que, além do preço reduzido, apresentou as melhores características, sendo combinado aos módulos leitor de cartão SD e Real Time Clock. E apresentado ainda nesse relatório, o procedimento para prototipagem e testes de modulo de aquisição e transmissão de dados proposto, bem como os algoritmos, hardwares utilizados para teste e validação e o layout de placa de circuito impresso.

Palavras-Chave: Dados Meteorológicos. Plataforma de Coleta de Dados. Wireless. ATMega. *Wi-Fi*. ESP-01.

LISTA DE ILUSTRAÇÕES

Pág.

LISTA DE TABELAS

Pág.

LISTA DE ABREVIATURAS E SIGLAS

bps	–	Bits por segundo
CI	–	Circuito Integrado
GDF	–	Grupo de Dispositivos Fotovoltaicos
GFSK	–	*Gaussian Frequency-Shift Keying*
GPIO	–	*General Porpose Input/Output*
HTTP	–	*Hypertext Transfer Protocol*
IDE	–	*Integrated Development Environment*
INPE	–	Instituto Nacional de Pesquisas Espaciais
IP	–	*Internet Protocol*
Kbps	–	Kilobits por segundo
m	–	Metros
Mbps	–	Megabits por segundo
ms	–	Milisegundos
MTU	–	*Maximum Transmission Unit*
P2P	–	*Peer-to-Peer*
PCD	–	Plataforma de Coleta de Dados
PCI	–	Placa de Circuito Impresso
RTC	–	*Real Time Clock*
SBCDA	–	Sistema Brasileiro de Coleta de Dados Ambientais
TCP	–	*Transmission Control Protocol*
Wi-Fi	–	*Wireless Fidelity*

SUMÁRIO

Pág.

1 INTRODUÇÃO

1.1 Motivação

O monitoramento e análise de dados meteorológicos são de extrema importância na sociedade, uma vez que previsões do tempo são essenciais para o funcionamento de diversos serviços como a aviação, que necessita destes dados sendo atualizados constantemente para manter a segurança durante os voos (Costa, 2008), as atividades agrícolas e na previsão do tempo, estudos sobre o clima ou na previsão de fenômenos naturais que afetam a saúde e a segurança humana (Giroto et al., 2015).

Além de medições e previsões do tempo, que se trata da análise de eventos de curto período, os dados meteorológicos também têm seu papel no auxílio de estudos das mais diversas áreas uma vez que diversas pesquisas necessitam de medições do ambiente. O Grupo de Dispositivos Fotovoltaicos - GDF, do Instituto Nacional de Pesquisas Espaciais - INPE, é um exemplo desse fato tendo em seu portifólio pesquisas que necessitam da constante medição da incidência da radiação solar (Oliveira et al. 2017), além de outros aspectos do ambiente.

Estes dados ambientais são coletados ao longo do tempo por uma plataforma de coleta de dados (PCD) devidamente posicionada em termos de coordenadas geográficas.

Porém o uso dessas plataformas de coleta conta com um grande empecilho: o custo. Atualmente muitos equipamentos utilizados nas PCDs são importados, fato que dificulta a aquisição de mais dispositivos para ampliação da rede principalmente pela frequente alta do dólar.

Outro problema desse sistema vem com o ambiente, uma vez que geralmente as plataformas se encontram em locais de difícil acesso. Atualmente, no projeto sobre dispositivos fotovoltaicos do GDF os dados são recuperados de forma manual, onde o dispositivo precisa ser conectado via cabos à um computador para que a aquisição possa ser feita. Esse segundo método pode ser contornado com a aquisição de um dispositivo de conversão wireless, que por sua vez também costuma ser de origem importada.

Com essa motivação, o GDF mantém um projeto de desenvolvimento de uma PCD de baixo custo e o objetivo principal desse projeto é aprimorar essa plataforma com a adição de um dispositivo de transmissão de dados sem fio capaz de facilitar o acesso aos dados coletados.

1.2 Objetivos

1.2.1 Objetivos Gerais

Os objetivos gerais desse trabalho consistiram no aprimoramento de uma plataforma dedicada para coleta de dados ambientas através de dispositivos capazes de estabelecer uma conexão sem fio para o envio de dados à um computador remoto além de armazenar *logs* de sensores conectados. Estes *logs* consistem nos dados a serem enviados de forma remota.

1.2.2 Objetivos Específicos

Prototipagem do módulo de aquisição e transmissão de dados para a PCD segundo os passos a seguir:

a) Pesquisa de dispositivos de transmissão de dados;
b) Teste de desempenho destes dispositivos;
c) Comparação entre as vantagens e desvantagens de cada abordagem;
d) Escolha da abordagem definitiva para o desenvolvimento do módulo;
e) Modelagem do circuito do módulo;
f) Montagem do protótipo;
g) Testes em laboratório;
h) Testes em campo.

2 INTRODUÇÃO TEÓRICA

2.1 Plataforma de Coleta de Dados

As chamadas Plataformas de Coleta de Dados (ou PCD) são dispositivos munidos de sensores automatizados capazes de ler o estado do ambiente em que estão inseridos. Esse estado é medido através dos mais diversos parâmetros dependendo do contexto em que os dados são utilizados, podendo ser desde sensores de velocidade e direção dos ventos até dispositivos capazes de medir a incidência dos espectros dos raios solares.

As PCDs podem ser divididas em duas categorias, terrestres ou orbitais (satélites), sendo a primeira o objeto de estudo deste projeto. Plataformas terrestres geralmente tem o formato de torre, como pode ser visto na Figura 1, onde seus sensores são instalados e conectados à um dispositivo coletor de dados. Esse dispositivo pode ser responsável tanto por apenas coletar os dados como também por transmiti-los.

Figura 1: Exemplo de PCD

Fonte: Portal das Missões. Disponível em: https://bityli.com/svfly.

Estas plataformas são instaladas em pontos estratégicos para melhor fidelidade dos dados, o que geralmente pode significar sua instalação em locais remotos ou de difícil acesso.

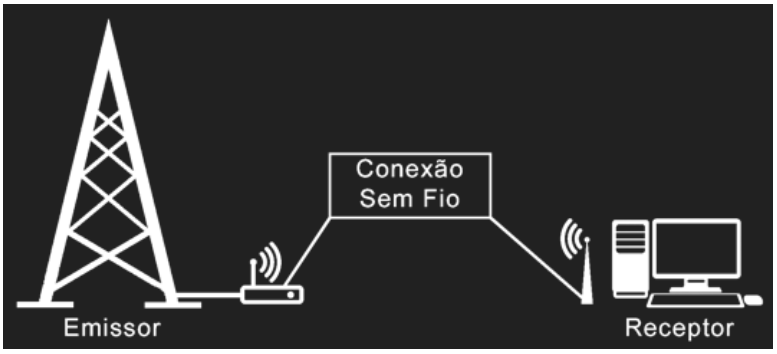
2.2 Transmissão Sem Fio

A transmissão sem fio consiste no envio de dados digitais por meio de ondas eletromagnéticas. O conceito de transmissão sem fio foi primeiramente introduzido pelo físico italiano Guglielmo Marconi ao demonstrar o funcionamento do telégrafo sem fio (Tanenbaum, 2003). Desde então este sistema vem crescendo cada vez mais de forma que hoje é a base de grande parte da arquitetura da *World Wide Web* e, para algumas pessoas, a única forma de se conectar à Internet.

A estrutura de uma rede sem fio pode variar muito dependendo da necessidade. Dentre as arquiteturas conhecidas de redes sem fio é possível citar algumas que serão abordadas nesse trabalho, sendo elas as redes *mesh* e redes *peer-to-peer* (P2P).

Redes P2P são as redes mais simples de serem entendidas, podendo consistir em um ou mais nós interligados ou não. Esse princípio de rede consiste em um grafo onde todos os nós podem se conectar e trocar informações sem o intermédio de um nó servidor. O modo mais básico de uma conexão P2P se trata de apenas dois nós interligados se comunicando. Esse nesse modo de operação cada nó recebe uma denominação dependendo de sua função no momento. Os nós podem ser denominados emissores, quando estão enviando dados, ou receptores, quando estão recebendo dados. Na Figura 2 é possível ver o exemplo de uma rede P2P simples aplicada à um sistema de aquisição de dados de uma PCD onde a plataforma é considerada emissora, pois coleta e envia os dados, e o dispositivo externo conectado é considerado receptor.

Figura 2: Topologia de um sistema sem fio Emissor-Receptor.

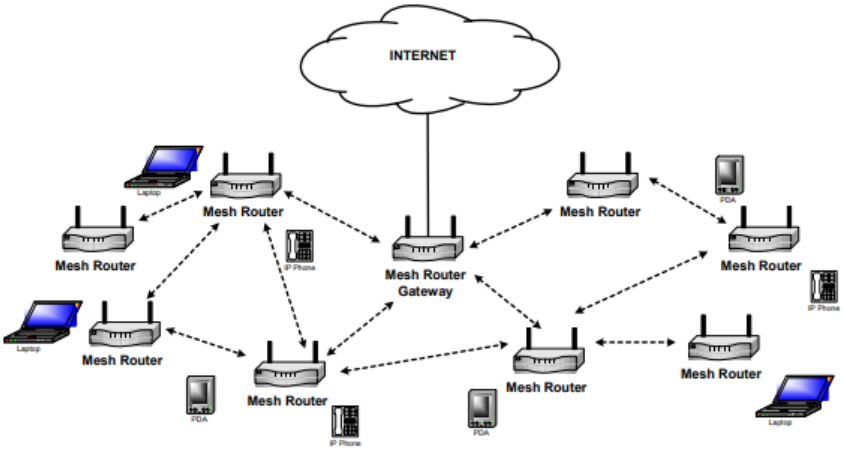


Fonte: O Autor.

A topologia de redes *mesh* se trata de uma abordagem mais complexa, comum na organização de redes sem fio. Este tipo de rede se difere da abordagem padrão das redes cabeadas onde normalmente em algum ponto a conexão é feita via um cabo central (*backbone*), neste caso o roteamento é feito de forma dinâmica, onde cada nó decide para onde enviar o pacote de dados recebido uma vez que é conectado a vários outros nós. (Abelém et al., 2007). Abaixo na Figura 3 é possível observar um exemplo de topologia *mesh*.

Neste tipo de rede é comum a existência de dois tipos de nós roteadores: os nós de acesso (*hotspots*) e os nós centrais. Os *hotspots* têm a função de conceder acesso à rede para dispositivos externos, enquanto os nós centrais têm a função de rotear os pacotes à fim de fazê-los chegarem ao seu destino.

Figura 3: Topologia de uma Rede *Mesh*.



Fonte: Abelém et al. (2007).

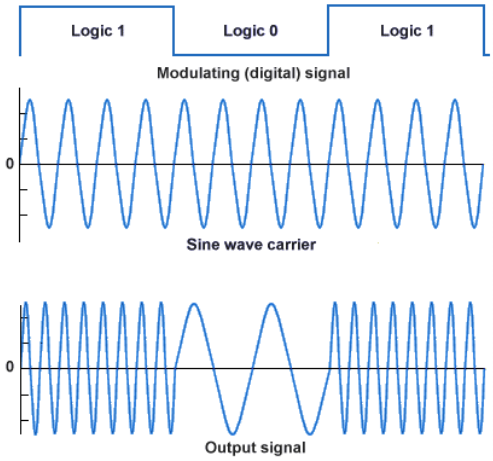
2.3 *Gaussian Frequency-Shift Keying*

A transmissão sem fio pode ser feita de diversas formas, porém todas elas têm um elemento em comum: o uso de ondas. O modo mais ortodoxo de se enviar dados por vias aéreas é a partir de ondas eletromagnéticas, e para isso foi desenvolvida uma técnica de

modulação de frequência (GFSK) onde, dependendo o valor digital do sinal a ser enviado, a frequência da onda é alterada para determinado valor (TechnologyUK, 2021).

De forma geral existe uma frequência base para o envio do dado, sendo essa frequência modulada, caso o dado tenha o valor 1, para um valor mais alto ou para um valor mais baixo, caso o dado tenha o valor 0. Abaixo na Figura 4 esse comportamento pode ser observado.

Figura 4: Comportamento de uma transmissão via GFSK



Fonte: TechnologyUK (2021)

1.4 Microcontroladores e ATMega 328

Assim como processadores, os microcontroladores podem ser classificados como o cérebro de um dispositivo eletrônico (SILBERSCHATZ et al., 2015). Tendo um tamanho reduzido quando comparados à um processador e também com menos capacidade de armazenamento e processamento, mas ainda sim portando periféricos como conversores A/D e D/A, módulos memória, etc., em seu encapsulamento, os microcontroladores são escolhas ideais para um projeto de pequeno porte onde é desejado praticidade e versatilidade. Microcontroladores também são altamente utilizados para prototipagem, uma vez que é possível programa-los e reprogramá-los quantas vezes for necessário, tornando fácil alterar seu comportamento.

Para uso como protótipo, geralmente os microcontroladores são acoplados à algumas plataformas generalistas para a facilidade de uso, acabando com a necessidade

da elaboração de uma placa de circuito impresso para cada projeto, tornando o microcontrolador mais genérico e possibilitando seu uso em diversos protótipos diferentes.

Dentre as mais diversas plataformas de prototipagem baseadas em microcontroladores a mais famosa é o Arduino, que se trata de um dispositivo modular de *hardware* livre projetado em torno de um microcontrolador da família AVR (como por exemplo ATMega328, ATTiny85, entre outros), cuja linguagem de programação aceita é uma versão adaptada do C++.

3 DESENVOLVIMENTO

3.1 Materiais e Métodos

Para a execução desse projeto alguns dispositivos foram utilizados à fim de testar a melhor forma de envio de dados por vias aéreas e comparar com o desempenho do *data logger* atualmente utilizado, o CR-1000 da Campbell Scientific. Estes materiais foram submetidos à testes de desempenho para tanto validar seu desempenho nominal fornecido pelo fabricante quanto para definir qual aparelho se encaixaria melhor às necessidades do projeto. Abaixo segue uma lista de todos os dispositivos de transmissão utilizados:

a) nRF27L01;
b) HC-06;
c) ESP-01;
d) NodeMCU.

Cada um desses dispositivos foi submetido à testes de taxa de transmissão, onde diversos pacotes de 32 bits foram enviados somando um total de 640 bits e seu tempo de envio total foi registrado para assim poder ser feita a relação de bits/tempo, testes de latência, onde o tempo de envio de cada pacote foi medido e ao fim uma média desse tempo foi feita a fim de estimar o tempo de resposta de cada dispositivo, e testes de distância máxima, onde pacotes pequenos (menos de 4 bytes) eram enviados e a maior distância sem perdas foi registrada. Sabendo que os módulos não apresentariam valores exatos e fixos para cada parâmetro, cada bateria de testes foi executada três vezes e o resultado final foi a média simples dos resultados parciais e para os casos em que a

variação das parciais foi superior à 50% da média, a classificação “Inconclusivo” foi atribuída.

Ao final dos testes um protótipo simplificado da PCD referente a cada um dos módulos utilizados foi idealizado a fim de entender sua complexidade, fazer um levantamento dos dispositivos necessários para cada um e por fim um orçar cada abordagem levando em conta apenas os itens variáveis de cada projeto, tendo como objetivo verificar qual das abordagens teria o menor preço.

A partir da escolha da abordagem final foi então desenvolvido o protótipo da plataforma e desenhada sua PCI com o auxílio do simulador e roteador de placas EasyEDA. Os componentes finais da plataforma foram:

a) Microcontrolador Atmega328 (Anexo 2);
b) Módulo ESP-01 (Anexo 3);
c) Módulo Leitor de Cartão SD 4MD36 (Anexo 4);
d) Módulo RTC DS3231 (Anexo5);
e) 2 resistores, sendo um de xxx Ω e outro de xxx Ω;
f) Fios para a ligação entre os dispositivos.

Além da implementação do *hardware* da PCD, também foi necessário o desenvolvimento do *software* a ser embarcado nos microcontroladores utilizados e presente no dispositivo receptor dos dados. Para a implementação do código a ser inserido nos controladores foi utilizada a linguagem de programação C++ nativa dos dispositivos, enquanto no dispositivo receptor a rotina de aquisição e armazenamento dos dados foi implementada em Python para todas as tecnologias de transmissão.

Em todos os testes iniciais o microcontrolador ATMega328 foi substituído por um Arduino Uno (que conta com o mesmo microcontrolador ATMega) para que alguns recursos de teste pudessem ser utilizados, como por exemplo o monitor serial.

3.2 CR-1000

O CR-1000 trata-se de um *data logger*, ou seja, um dispositivo capaz de registrar o valor de uma série de sinais, guardá-los em *log* e transmiti-los via comunicação serial pelo protocolo RS-232. Esse dispositivo foi desenvolvido pela empresa Campbell

Scientifc e atualmente encontra-se fora de linha. Segundo seus dados nominais, suas métricas de desempenho são as seguintes:

Tabela 1: Desempenho do CR-1000

Taxa Máxima	Aprox. 20 Kbps
Latência	Não informado
Distância Máxima	60 metros

Fonte: Pinheiro (2011).

3.3 nRF24L01

Este é um dispositivo modular compatível ao ATMega 328 capaz de se comunicar via rede sem fio. Esse dispositivo utiliza de um protocolo próprio de envio de mensagens via GFSK podendo se comunicar apenas com dispositivos do mesmo modelo, sendo então necessário o desenvolvimento de um módulo emissor e um módulo receptor. Esse módulo tem algumas limitações quanto ao seu uso como por exemplo não ser capaz de enviar um pacote de dados maior que 32 bits e caso exceda esse limite o pacote é apenas descartado.

A troca de informações entre esse módulo e o microcontrolador se dá via comunicação SPI, utilizando os pinos de entrada e saída do ATMega próprios para essa finalidade.

Abaixo na Tabela 2 estão presentes os dados de desempenho do módulo fornecidos pelo fabricante.

Tabela 2: Desempenho do nRF24L01

Taxa Máxima	1 Mbps
Latência	Não informado
Distância Máxima	1000 metros

Fonte: Nordic Semiconductor (2006).

3.4 HC-06

Se tratando de um módulo de comunicação Bluetooth bem comum de se encontrar em conjunto com um Arduino, o HC-06 foi a segunda escolha de tecnologia de transmissão para esse projeto. Tendo a vantagem de se comunicar com qualquer outro aparelho que tenha a tecnologia Bluetooth, esse dispositivo se comunica com o ATMega via comunicação serial padrão, utilizando suas entradas Rx e Tx.

Este dispositivo conta com uma limitação de 251 bytes para carga útil de pacote, porém caso exceda o pacote é automaticamente ajustado e enviado na forma de dois ou mais pacotes. O dispositivo não conta com um sistema de controle de perdas, portanto é necessário implementar manualmente um sistema de controle de integridade dos pacotes a fim de lidar com dados perdidos.

Abaixo é possível encontrar a Tabela 3 com os dados nominais de desempenho informados pelo fabricante.

Tabela 3: Desempenho do HC-06

Taxa Máxima	1 Mbps
Latência	Não informado
Distância Máxima	1000 metros

Fonte: Guangzhou HC Information Technology Co. (2006).

3.5 ESP-01 e NodeMCU

Estes módulos se tratam de plataformas com microcontroladores programáveis, assim como o Arduino, tendo o mesmo microcontrolador como base (ESP8266) variando apenas em seu tamanho e capacidade. O ESP-01 se trata de um dispositivo de tamanho reduzido, com poucas entradas para GPIO sendo geralmente utilizado em paralelo com um Arduino ou um ATMega 328 *standalone* através da comunicação serial. Já o NodeMCU se trata de um encapsulamento mais robusto do ESP8266 podendo até substituir o Arduino em seu uso, uma vez que o mesmo tem até mais pinos de GPIO do que alguns dispositivos da família portadora do ATMega.

Tratando-se agora de utilidade, o ESP8266 possibilita ao dispositivo ao qual está embarcado se conectar à uma rede Wi-Fi padrão e trocar informações com qualquer outro dispositivo conectado nessa mesma rede já que tem a capacidade de funcionar como um servidor *Web* que executa o protocolo HTTP, protocolo que permite a troca de arquivos de texto entre dispositivos via Internet.

Assim como os outros dispositivos, as plataformas portadoras do ESP8266 têm uma limitação para a transferência de pacotes via rede, limitação essa que se dá devido ao fato do tamanho máximo de pacote ser relacionado ao MTU da rede em que está conectado, que geralmente é de 1500 bytes. Para o caso de estouro do limite do pacote, o mesmo é automaticamente redimensionado em sub pacotes. Quanto à integridade do pacote também existe um mecanismo de controle de recebimento dos pacotes, uma vez que o microcontrolador implementa a pilha de protocolos de controle de rede TCP/IP (Espressif Systems, 2021).

É fornecido pelo fabricante a tecnologia Wi-Fi suportada pelo módulo, portanto é possível saber alguns dados de seu desempenho, que estão apresentados abaixo na Tabela 4.

Tabela 4: Desempenho do ESP8266

Taxa Máxima	150 Mbps
Latência	Não informado
Distância Máxima	180 metros

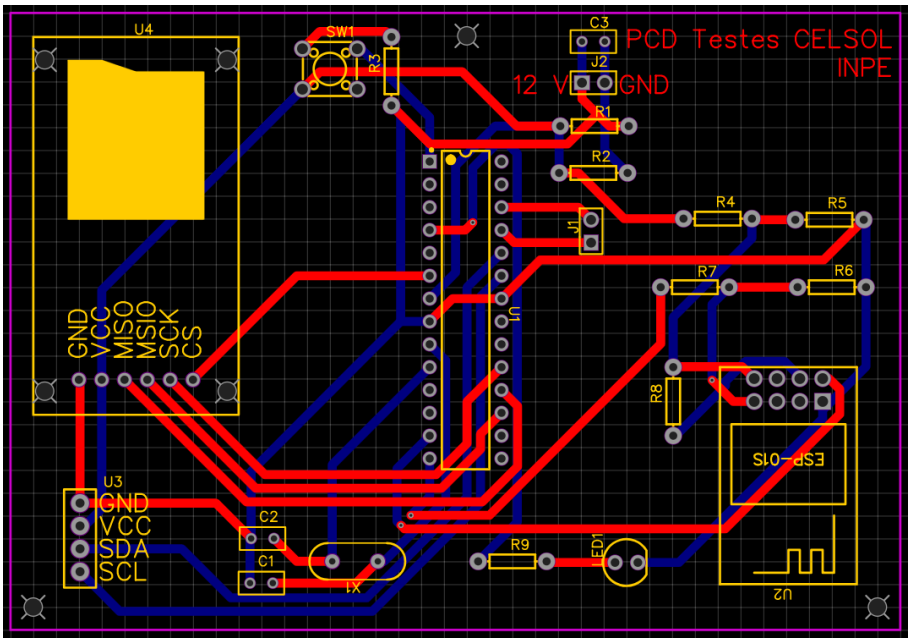
Fonte: Espressif Systems (2021).

3.6 Desenvolvimento do Módulo de Aquisição

Após a escolha tecnologia de transmissão foi possível desenvolver o modelo da plataforma, unindo todos os módulos necessários em apenas um circuito. O modelo esquemático pode ser visto ao final deste relatório no Anexo 1.

Desenvolvido o modelo esquemático do circuito, com o auxílio da ferramenta de roteamento de placas do *software* EasyEDA, a placa de circuito impresso foi desenhada possibilitando a produção futura do protótipo da PCD. Acima na Figura 5 é possível ver uma representação da placa final do protótipo.

Figura 5: Representação da Placa de Circuito Impresso da PCD.



Fonte: O Autor.

Para os primeiros testes foi utilizado um módulo DHT-11 para medir a temperatura do ambiente e fazer as correções necessárias nos códigos base dos controladores a fim de se conseguir uma estabilidade do sistema, e em seguida seria utilizado um radiômetro a fim de realizar os testes finais do dispositivo através da medição das diversas leituras fornecidas pelo mesmo (UVA, UVB, Irradiância e Temperatura).

Outro passo do desenvolvimento da plataforma foi a implementação dos algoritmos a serem embarcados nos controladores. Para facilitar correções ou alterações o código aplicado ao ATMega foi dividido em módulos, sendo cada módulo um arquivo diferente. Esses módulos foram separados da seguinte forma:

1. **Rotina principal:** Arquivo que interliga todos os outros módulos, além de contar com as funções *setup* e *loop* necessárias para o funcionamento controlador (Anexo 6);

2. **Controle do relógio:** Arquivo que conta com funções para a inicialização, leitura e tratamento dos dados obtidos através do módulo RTC (Anexo 9);
3. **Controle de dados:** Arquivo com funções para o armazenamento e leitura de dados no cartão SD além de ser responsável por enviar os mesmos para o módulo ESP-01 quando requisitado (Anexo 8);
4. **Controle do Sensor de temperatura:** Arquivo desenvolvido para inicializar, fazer leituras e tratar os dados fornecidos pelo sensor DHT-11 (Anexo 10);
5. **Arquivo de constantes e variáveis globais:** Um arquivo para reunir as constantes e as variáveis utilizadas de forma comum por todos os módulos como por exemplo os pinos de conexão de cada dispositivo, estruturas de dados armazenar dados de leitura, entre outras (Anexo 7).

A leitura dos módulos externos foi feita através de bibliotecas públicas importadas no código, sendo elas: SPI e RTClib para leitura do relógio, SD para gerenciamento do cartão, SoftwareSerial para comunicação entre o ATMega e o ESP-01 e DHT para leitura do sensor de temperatura.

Já em relação ao código aplicado ao controlador ESP-01, o mesmo foi implementado em apenas um arquivo (Anexo 11) que é responsável por inicializar o servidor *web* do módulo através das bibliotecas nativas ESP8266WiFi, WiFiClient e ESP8266WebServer. Foi definido um endereço de IP fixo para o módulo possibilitando um fácil acesso ao mesmo, uma vez que desde o início de sua execução esse endereço já é conhecido. Caso isso não fosse feito seria necessário o auxílio de um gerenciador de dispositivos para descobrir seu IP a cada conexão em uma rede diferente. Dessa forma então o servidor do dispositivo pode ser acessado pelo endereço *http://192.168.0.100/*. É importante ressaltar que esse endereço de IP deve ser adaptado caso o endereço de rede interna do roteador não siga o padrão 196.168.0.1.

O código embarcado no módulo também é responsável por gerenciar o acesso à página raiz do servidor, ou seja, o endereço padrão *http://192.168.0.100/* que apresenta dados do dispositivo como rede conectada, endereço IP e endereço MAC¸ o *endpoint http://192.168.0.100/raw* que se trata do endereço utilizado para requisitar a transferência de dados e a caso de página não encontrada, onde o servidor retorna a mensagem “File Not Found”, a URL acessada, o tipo de requisição, os argumentos e o *status* HTTP 404.

Após o desenvolvimento da estrutura do módulo, os primeiros testes foram feitos por partes, ou seja, o projeto for separado em submódulos que foram testadas de maneira independente simulando o retorno das outras partes não conectadas para que se pudesse verificar se as conexões entre os módulos foram feitas de forma correta. Os testes dos módulos foram feitos com os mesmos conectados ao Arduino no lugar do ATMega328.

Os testes foram feitos na seguinte ordem:

1. *Upload* de códigos no ATMega328;
2. Leitura do módulo RTC;
3. Leitura e escrita no cartão SD;
4. Requisições HTTP entre ESP-01 e dispositivo de aquisição via Python;
5. Leitura do sensor DHT e escrita no cartão SD com estampa de data e hora;
6. Comunicação entre Arduino e ESP-01.

2.7 Aquisição dos Dados

O sistema de aquisição de dados, como comentado na Seção 2.1, foi desenvolvido em na linguagem Python utilizando algumas bibliotecas externas. Para os testes iniciais, nenhuma interface gráfica foi implementada e apenas as bibliotecas Serial, Serial.Tools, Requests e Time foram utilizadas, sendo as duas primeiras para os testes com os módulos nRF24L01 e HC-06, a terceira para os módulos NodeMCU e ESP-01 e a última sendo utilizada em todos os testes. Nessa etapa do projeto o algoritmo apenas fazia requisições simples de dados e contava o tempo de envio.

Após a escolha da abordagem final o código foi refinado (Anexo 12) e a biblioteca PySimpleGUI (Anexos 13 e 14) foi adicionada com a finalidade de se criar uma interface gráfica mais intuitiva para o usuário final. Nesse ponto do projeto foi necessário implementar um tratamento mais refinado de requisições e tratamento de dados para lidar com a "conversa" entre o *Web Server* e o dispositivo receptor, uma vez que o módulo ESP-01 envia os dados na forma de uma *String* que contêm o valor diversas leituras salvas no cartão SD separadas por virgula. O algoritmo então separa cada leitura por data e salva linha por linha em um arquivo de extensão *.txt*.

Como medida de segurança, foi implementado um sistema de login simples no sistema de aquisição apenas para evitar acessos indevidos à aplicação.

4 RESULTADOS E DISCUSSÃO

4.1 Resultados dos Testes Comparativos

Os testes de desempenho foram feitos na ordem apresentada no Capítulo 2, porém alguns problemas foram observados. Estes problemas serão explicados mais à frente. Abaixo na Tabela 5 é possível observar o resultado dos testes para cada módulo.

Tabela 5: Resultados dos testes de desempenho

	nRF24L01	**HC-06**	**ESP-01/NodeMCU**
Transmissão	0,48 Kbps	2,4 Kbps	Inconclusivo
Latência	Não Testado	99 ms	Inconclusivo
Distância	Não Testado	8 m	70 m

Fonte: O Autor

Inicialmente analisando o módulo nRF24L01 não foi possível testar dois dos três parâmetros de desempenho devido à uma falha no dispositivo, fato que também influenciou no desempenho do primeiro teste, que apresentou um resultado extremamente abaixo do resultado nominal.

O segundo módulo analisado foi o HC-06, com a tecnologia Bluetooth. Foi primeiramente testado sem nenhum mecanismo de tratamento de perdas, porém foi observada uma grande taxa de perdas para dados extensos. Portanto os testes foram refeitos com um mecanismo simples para impedir as perdas que consistia em uma simples espera entre o envio dos pacotes, o que impedia que um conjunto de dados fosse enviado antes que outro tenha sido recebido. Com esse mecanismo houve uma grande perda de desempenho fazendo com que o módulo também apresentasse uma taxa de transmissão muito menor que a taxa nominal. No mais as outras métricas de desempenho apresentaram um resultado satisfatório.

Por fim os módulos ESP-01 e NodeMCU, por portarem o mesmo microcontrolador, foram submetidos aos testes em conjunto em diversas redes Wi-Fi. As métricas de taxa de transmissão e de latência apresentaram dados inconclusivos. Isso se

dá devido ao fato de o módulo estar sujeito ao desempenho da rede em que está conectado. Apesar desse resultado inconclusivo o dispositivo mostrou uma grande fidelidade ao desempenho da rede conectada. A métrica de distância foi a única que apresentou resultados consistentes, porém por se tratar de uma distância muito extensa não foi possível realizar o teste em um ambiente em que não houvessem obstáculos, fato que fez com que o resultado ficasse um pouco abaixo do esperado.

Ponderando estes testes e alguns outros fatores mais subjetivos foi decidido que a resolução que mais se encaixava no escopo do projeto é a que empregava o módulo ESP-01 em conjunto com o ATMega 328, pela sua satisfatória taxa e distância de transmissão já contanto com mecanismos de controle de perdas, a possibilidade de fácil inserção de outros módulos à configuração (como por exemplo o módulo RTC ou o leitor de cartão SD) e seu custo reduzido em comparação aos outros módulos.

4.2 Teste Modulares

Na fase de testes em submódulos, a primeira etapa obteve um resultado inconsistente uma vez que em alguns momentos não era possível carregar o código alvo no microcontrolador. Todas as 5 etapas seguintes tiveram resultados positivos indicando que as conexões foram definidas de forma correta.

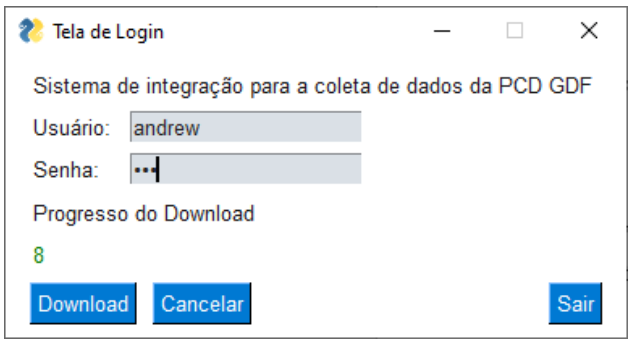
4.3 Aplicativo de Aquisição

Com o auxílio da biblioteca PySimpleGUI, como citado na Seção 2.7, foi possível tornar a aquisição de dados da PCD mais intuitiva para todo os usuários ao mesmo tempo em que as rotinas de requisição ficavam mais robustas.

A aquisição foi feita de forma que ao início da execução a PCD deveria informar a quantidade de itens salvos para que assim a aplicação possa fazer a requisições dos pacotes de dados. Para os testes iniciais foi configurado para que apenas as leituras com a mesma etiqueta de data e hora fossem enviadas, o que configura uma linha do arquivo salvo no cartão SD. Esse número também é utilizado para atualizar uma barra de progresso que é apresentada na interface da aplicação para o usuário.

Abaixo na Figura 6 pode ser vista a interface gráfica da aplicação durante um teste com requisições simuladas.

Figura 6: Teste da aplicação para aquisição de dados

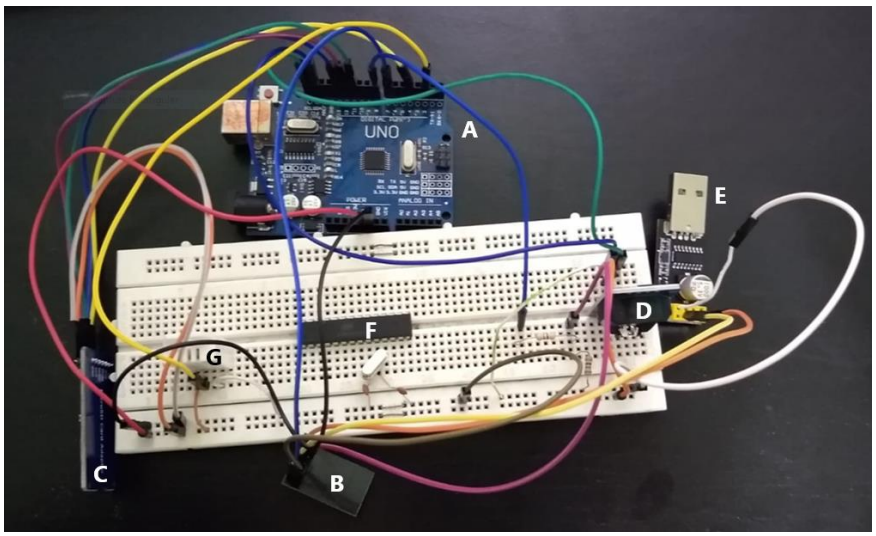


Fonte: O Autor

4.4 Testes na *Protoboard*

Essa segunda fase de testes foi realizada com a montagem da PCD em uma *protoboard* com a conexão de todos os módulos com a finalidade de testar os códigos desenvolvidos e embarcados nos microcontroladores. Abaixo na Figura 7 é possível ver a montagem da mesma.

Figura 7: Montagem em *protoboard* da PCD.



Fonte: O Autor.

Como é possível ver na figura acima, o CI do ATMega328 (F) está presente na montagem assim como um Arduino Uno (A), isso se dá devido ao fato dessa fase de testes ter sido dividida em duas etapas: Testes com Arduino e testes com ATMega328. Na imagem também podem ser visualizados os módulos ESP-01 (B), Leitor SD (C), RTC (D), Adaptador USB para alimentação do ESP (E) e sensor DHT (G).

Também é possível identificar um módulo USB à direita da imagem. A inclusão desse módulo foi necessária para manter o funcionamento do módulo ESP-01, pelo fato da saída de 3,3 V do Arduino não ser capaz de manter o fornecimento de corrente necessário para alimentar o módulo ESP. Dessa forma o adaptador USB utilizado para carregar programas no ESP-01 foi utilizado para alimentar o mesmo.

Para a conexão entre os dispositivos foi criada uma rede Wi-Fi privada com apenas ambos emissor e receptor conectados.

Esta fase de testes também apresentou resultados satisfatórios pois a plataforma demonstrou o comportamento esperado. A partir de análises do conteúdo salvo no cartão SD e de mensagens de progresso apresentadas no monitor serial do Arduino foi possível verificar que os sistemas de gerenciamento do cartão SD e leitura dos sensores funcionava da maneira esperada, assim como a comunicação ESP-Arduino. O servidor *Web* criado também apresentou o comportamento esperado visto que foi criado com sucesso, manter o comportamento esperado para a página raiz como pode ser visto abaixo na Figura 8 e ao se acessar o *endpoint* "/raw" apresentava uma leitura do cartão SD para ser recuperada pelo sistema de aquisição. Analisando conferindo os dados do registro de recebimento do sistema de aquisição e também do arquivo salvo no computador que realizou a aquisição dos dados foi possível confirmar a validade dos mesmo, estando conforme o que foi salvo no cartão SD.

Figura 8: Página raiz do servidor *web.*

Fonte: O Autor

Os testes com a substituição do Arduino pelo ATMega328 não puderam ser realizados pois, como citado na Seção 3.2, o processo de *upload* do código para o microcontrolador apresentou um comportamento inconsistente fazendo com que o algoritmo não fosse carregado da forma correta.

5 CONCLUSÃO

A pandemia de Covid-19 durante o desenvolvimento do projeto dificultou o prosseguimento desse projeto devido à paralização de serviços e o acesso aos laboratórios do INPE. Esse fato também não permitiu a confecção da placa de circuito impresso projetada para a plataforma devido à indisponibilização da plataforma de prototipagem de PCI do INPE.

A pesquisa bibliográfica e análises do protótipo do sistema de coleta de dados desenvolvido mostraram que ele atende os requisitos do projeto para ser utilizado em testes de campo, porém esses testes não puderam ser realizados devido as restrições impostas pela pandemia e a falta de tempo.

Os próximos passos do projeto incluem a revisão da comunicação Arduino-ESP e da rotina de criação da página "/raw", a fim de otimizar à aquisição de dados da plataforma e realizar testes relacionados à rotina de leitura de dados do radiômetro.

Também é necessário a implementação de mais tratativas de segurança, tanto em questão à ciberataques quanto prevenção de erros em caso de queda de conexão, uma vez que o sistema tem uma limitação de processamento o que o torna muito vulnerável à ataques do tipo *Denial of Service*.

Alterações no algoritmo embarcado no ATMega328 e na PCI do módulo também devem ser feitas à fim de permitir a conexão de sensores através de entradas analógicas multiplexadas.

Por fim, como proposta de trabalho futuro, é preciso finalizar o desenvolvimento da placa de circuito impresso, realizar a montagem dos componentes e iniciar os testes em laboratório e em campo da unidade de coleta de dados.

REFERÊNCIAS

COSTA, M. M. G. A. **Meteorologia Aeronáutica no Aeroporto de Guarulhos**. 7., 2008, Rio de Janeiro. Anais do VII Simpósio de Transporte Aéreo, p. 539-550. Disponível em: https://cabecadepapel.com/sites/viisitraer2008/pdf/500.pdf. Acesso em 19 mar. 2021.

GIROTO, D. B.; GULDONI, B.; TOMMASELLI, J. T. G. **A escola na estação meteorológica: A Importância da Meteorologia no Cotidiano Humano**. 2015., Presidente Prudente. Anais do 8º Congresso de extensão universitária da UNESP, p. 1-11, Disponível em: http://hdl.handle.net/11449/142284. Acesso em 19 mar. 2021.

OLIVEIRA, B. C. B.; VILELA, W. A.; NEVES, G. M.; PASIM, D. G. **Desenvolvimento de rotinas computacionais para o processamento de dados espectrais da radiação solar.** São José dos Campos: INPE, 2017. 30 p. Bolsa PIBIC/INPE/CNPq. IBI: <8JMKD3MGP3W34R/42KFT2P>. Disponível em: http://urlib.net/rep/8JMKD3MGP3W34R/42KFT2P. Acesso em 30 jul. 2021.

TANENBAUM, A. S. **Redes de Computadores**. 4 ed. Rio de Janeiro: Editora Campus, 2003. p. 23.

NSTITUTO NACIONAL DE PESQUISAS ESPACIAIS. **Sistema de Coleta de Dados**. 06 dez. 2019. Disponível em: http://www.cbers.inpe.br/sobre/coleta_dados.php. Acesso em 19 mar. 2021.

ABELÉM, A. J. G. et al. **Redes Mesh: mobilidade, qualidade de serviço e comunicação em grupo.** 2007. Disponível em: http://www.ic.uff.br/~celio/papers/minicurso-sbrc07.pdf. Acesso em 19 mar. 2021.

TECHNOLOGYUK. **Frequency Shift Keying**. 2021. Disponível em: http://www.technologyuk.net/telecommunications/telecom-principles/frequency-shift-keying.shtml. Acesso em: 26 jan. 2021.

SILBERSCHATZ, A.; GALVIN, P. B.; GAGNE, G. **Fundamentos de sistemas operacionais.** 9 ed. [S.l.]: LTC, 2015. ISBN 978-85-216-3000-5.

PINHEIRO, G. **A Interface Serial e o Padrão RS-232**. 2011. Disponível em: http://www.lee.eng.uerj.br/~gil/filas/Padrao%20RS-232.pdf. Acesso em: 26 jan. 2021.

NORDIC SEMICONDUCTOR. **NRF24L01 Datasheet**. 2006. 39p. Disponível em: https://www.sparkfun.com/datasheets/Components/nRF24L01_prelim_prod_spec_1_2.pdf. Acesso em: 26 jan. 2021.

GUANGZHOU HC INFORMATION TECHNOLOGY CO. LTD. **HC-06 Datasheet.** Disponível em: https://www.olimex.com/Products/Components/RF/BLUETOOTH-SERIAL-HC-06/resources/hc06.pdf. 2006. 17p. Acesso em: 26 jan. 2021.

ESPRESSIF SYSTEMS. **ESP32 Series Datasheet**. 2021. 65p. Disponível em: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf . Acesso em: 26 jan. 2021.

AI-THINKER CO. LTD. **Datasheet ESP-01**.v 2015 Disponível em: https://pdf1.alldatasheet.com/datasheet-pdf/view/1179098/ETC2/ESP-01.html. Acesso em 11 ago. 2021.

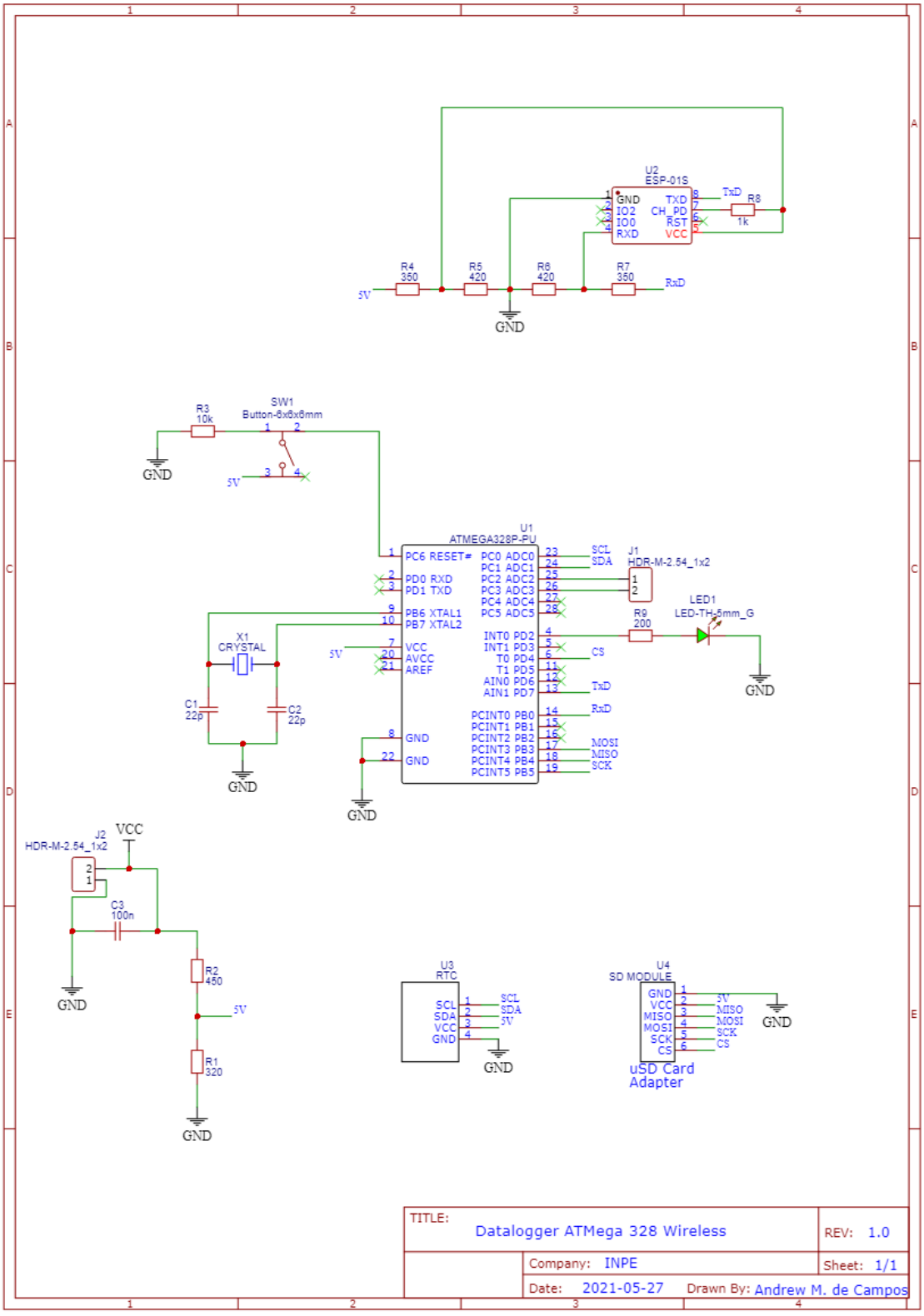
REITZ, K. ET AL. **Requests: HTTP for Humans™**. 2019 Disponível em: https://docs.python-requests.org/en/master/. Acesso em 17 ago. 2021.

PySimpleGUI™. **Python GUIs for Humans**. Disponível em: https://pysimplegui.readthedocs.io/en/latest/. Acesso em 17 ago. 2021.

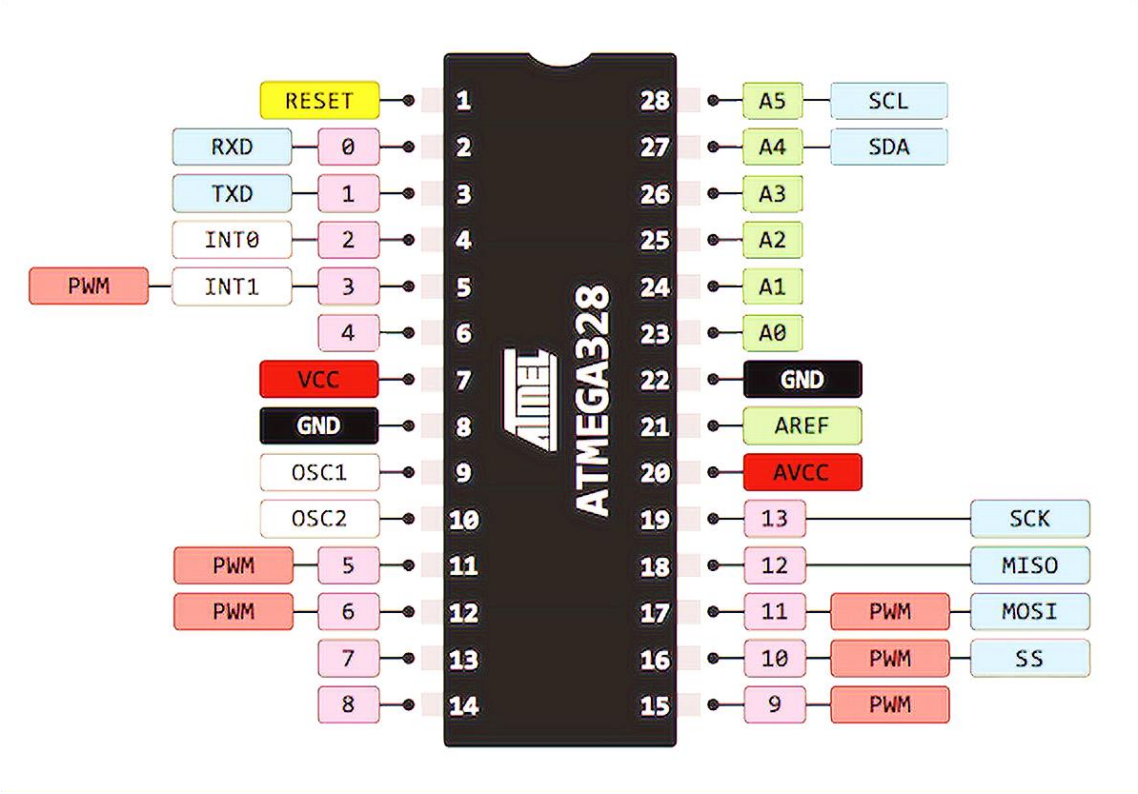
ARDUINO. **Arduino Docs**. 2021. Disponível em: https://docs.arduino.cc/. Acesso em: 17 ago. 2021.

ANEXOS

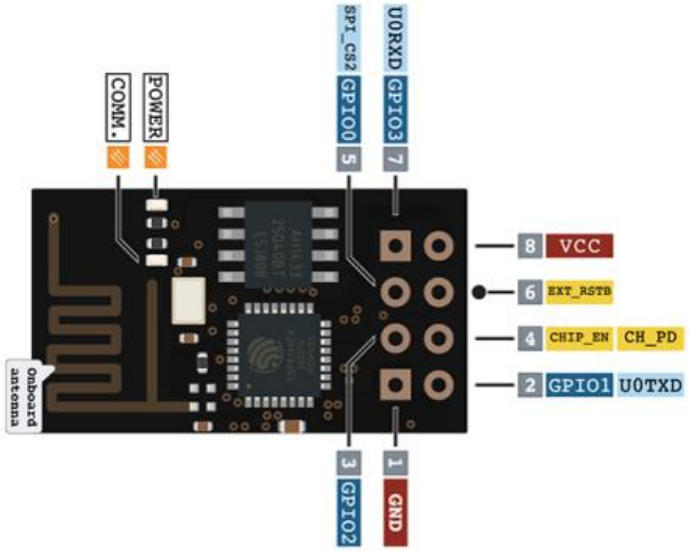
Anexo 1



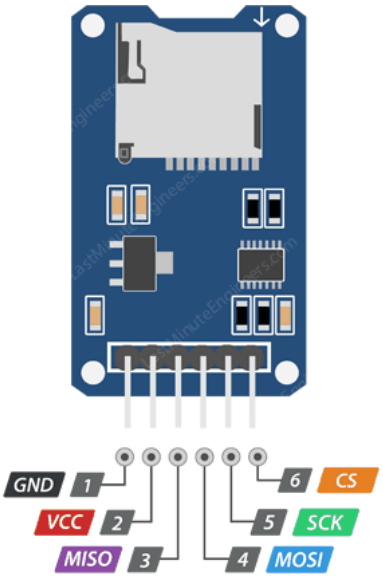
Anexo 2



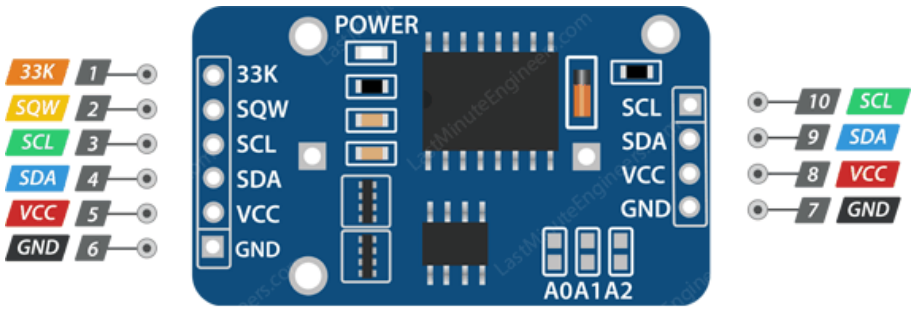
Anexo 3



Anexo 4



Anexo 5



Anexo 6

```
#include "globals.h"

rtc_data_t ultima, atual;
bool leitura_realizada = true;

void setup() {
    ErroNaRede = true;
    // Setup para Arduino
    pinMode(LED, OUTPUT);
    Serial.begin(9600);

    InicializarDht();
    InicializarRTC();
    InicializarSD();
    esp01.begin(9600);
    Serial.println("\nSistema Inicializado! " + RtcToString(LerRTC()));
    ultima = LerRTC();
}

void RealizarLeituras() {
```

```
    String leitura;
    leitura = RtcToString(LerRTC()) + ",";
    leitura += LerDHT();
    SalvarLog(leitura);
    Serial.println(leitura);
}

void loop() {
    char op = 'x';
    char esp_msg = '0';

    if (Serial.available()) {
        op = Serial.read();
    }

    if (esp01.available()) {
        esp_msg = esp01.read();
    }

    // Operacao manual do sistema
    switch (op) {
        case '0':
            RealizarLeituras();
            break;
        case 'l':
            LerArquivoLog();
            break;
        case 'p':
            Serial.println(F("Sistema Pausado"));
            while(!Serial.available());
            break;
        default:
            break;
    }

    switch (esp_msg) {
        case 't':
          Serial.println(F("Enviando dados ao ESP-01!"));
            EnviarRegistros();
            break;
        default:
            break;
    }

    atual = LerRTC();
    if (atual.minutos % 5 == 0 && !leitura_realizada) {
        RealizarLeituras();
        leitura_realizada = true;
    } else if (atual.minutos % 5 != 0){
        leitura_realizada = false;
    }
}
```

Anexo 7

```
#include <DHT.h>
#include <RTClib.h>
#include <SPI.h>
#include <SD.h>
#include <SoftwareSerial.h>

// DHT (Temperatura)
#define DHTPIN 2
#define DHTTYPE DHT22
// SD
#define CHIP_SELECT_SD 4
// ESP-01
#define RxD 7
#define TxD 8
// Referencias do arduino
#define LED 13
#define HIGH_PIN A0
#define LOW_PIN A1
#define ADC_RES 1024.0
#define AREF 5.0

// Estruturas de aquisicao
typedef struct {
    float umidade;
    float temperatura;
} dht_data_t;

typedef struct {
    int dia;
    int mes;
    int ano;
    int horas;
    int minutos;
    int segundos;
} rtc_data_t;

// Modulos
const DHT dht(DHTPIN, DHTTYPE);
const RTC_DS3231 rtc;
const SoftwareSerial esp01(RxD,TxD);

// Variaveis de controle
const char diasDaSemana[7][12] = {"Domingo", "Segunda-Feira", "Terça-
Feira", "Quarta-Feira", "Quinta-Feira", "Sexta-Feira", "Sábado"};
bool ErroNaRede;
```

Anexo 8

```
bool primeiro_envio = true;

void InicializarSD(){
     while (!SD.begin(CHIP_SELECT_SD)) {
```

```
        delay(2000);
    }
}

void SalvarLog(String LOG) {
    File Ambiente = SD.open("pcd.log",FILE_WRITE);

    if (Ambiente) {
        Ambiente.println(LOG);
    } else {
        Serial.println(F("Erro ao abrir 'pcd.log'"));
    }
    Ambiente.close();
}

void LerArquivoLog() {
    File Ambiente = SD.open("pcd.log");
    if (Ambiente) {
        while (Ambiente.available()) {
            Serial.write(Ambiente.read());
        }

    } else {
        Serial.println(F("Erro na leitura do arquivo 'pcd.log'"));
    }
    Ambiente.close();
}

void EnviarRegistros() {
    File Ambiente = SD.open("pcd.log");
    String linha;

    if (Ambiente) {
        while (Ambiente.available()){
            if (primeiro_envio) {
                linha = Ambiente.readStringUntil('\n');
                esp01.print(linha);
                Serial.println(linha);
                primeiro_envio = false;
                continue;
            }

            if (esp01.available()) {
                if (esp01.read() == 'x') {
                    Serial.println(F("Transmissão cancelada!"));
                    break;
                } else {
                    linha = Ambiente.readStringUntil('\n');
                    esp01.print(linha);
                    Serial.println(linha);
                }
            }
        }
    } else {
        Serial.println(F("Erro na leitura do arquivo 'pcd.log'"));
```

```
    }
    Ambiente.close();
}
```

Anexo 9

```
void InicializarRTC() {
    while (!rtc.begin()) {
        delay(2000);
    }

    if(rtc.lostPower()){
        Serial.println("RTC ficou sem energia. Redefinindo data...");
        // Ajusta data e hora
        rtc.adjust(DateTime(2021,8,14,19,42,0));
    }
}

String RtcToString(rtc_data_t leitura) {
    return String(leitura.dia) + "/" + String(leitura.mes) + "/" + String
(leitura.ano) + " " +
        String(leitura.horas) + ":" + String(leitura.minutos) + ":" + Str
ing(leitura.segundos);
}

rtc_data_t LerRTC() {
    rtc_data_t dadosRtc;
    DateTime now = rtc.now();

    dadosRtc.dia = now.day();
    dadosRtc.mes = now.month();
    dadosRtc.ano = now.year();
    dadosRtc.horas = now.hour();
    dadosRtc.minutos = now.minute();
    dadosRtc.segundos = now.second();

    return dadosRtc;
}
```

Anexo 10

```
void InicializarDht() {
    dht_data_t primeiraLeitura;
    // inicializa sensor
    dht.begin();
    // faz a primeira leitura
    primeiraLeitura.umidade = dht.readHumidity();
    primeiraLeitura.temperatura = dht.readTemperature();
    // verifica integridade da leitura
    while (isnan(primeiraLeitura.umidade) || isnan(primeiraLeitura.temper
atura)) {
        delay(2000);
        dht.begin();
        primeiraLeitura.umidade = dht.readHumidity();
```

```
        primeiraLeitura.temperatura = dht.readTemperature();
    }
}

String DhtToString(dht_data_t leitura) {
    return "Temperatura: " + String(leitura.temperatura) + " ºC | Umidade
: "
        + String(leitura.umidade) + " %";
}

String LerDHT( ) {
    dht_data_t dadosdht;
    dadosdht.umidade = dht.readHumidity();
    dadosdht.temperatura = dht.readTemperature();

    // Verifica se a leitura é válida
    if (isnan(dadosdht.umidade) || isnan(dadosdht.temperatura)) {
        return "Temperatura: -- ºC | Umidade: -- %";
    } else {
        return DhtToString(dadosdht);
    }
}
```

Anexo 11

```
#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266WebServer.h>

const char* ssid = "REPEST";
const char* password = "duendedoido";
ESP8266WebServer server(80);
int IP[4] = {192,168,15,100};

void requestData() {
    Serial.write('t');
    while (!Serial.available());
    server.send(200, "text/plain", Serial.readString());
}

void handleCancel() {
    Serial.write('x');
}

void handleRoot() {
    String root = "<!DOCTYPE html>";
    root += "<head><title>PCD Celsol</title><meta charset=\"UTF-
8\"><style>.all{font-family:";
    root += "'Gill Sans', 'Gill Sans MT', Calibri, 'Trebuchet MS', sans-
serif;}</style></head>";
    root += "<body><p class=\"all\"> <span style=\"font-
size: 30px;\"><strong>PCD Celol 1 - ";
    root += "Teste</strong></span><br>Sucesso ao se conectar à PCD. Infor
mações Gerais:<br>";
    root += "<strong>Rede: </strong>";
```

```
    root += ssid;
    root += "<br>";
    root += "<strong>IP: </strong>";
    root += String(WiFi.localIP()[0]) + "." + String(WiFi.localIP()[1]) +
 "." + String(WiFi.localIP()[2]) + "." + String(WiFi.localIP()[3]) ;
    root += "<br>";
    root += "<strong>MAC: </strong>";
    root += WiFi.macAddress();
    root += "</p>";
    root += "</body>";
    server.send(200, "text/html", root);
}

void handleNotFound() {
    String message = "File Not Found\n\n";
    message += "URL: ";
    message += server.uri();
    message += "\nMethod: ";
    message += (server.method() == HTTP_GET) ? "GET" : "POST";
    message += "\nArguments: ";
    message += server.args();
    message += "\n";

    for (uint8_t i = 0; i < server.args(); i++) {
        message += " " + server.argName(i) + ": " + server.arg(i) + "\n";
    }

    server.send(404, "text/plain", message);
}

void setup(void) {
    Serial.begin(9600);
    WiFi.mode(WIFI_STA);
    IPAddress ip(IP[0], IP[1], IP[2], IP[3]);
    IPAddress gateway(IP[0], IP[1], IP[2], 1);
    IPAddress subnet(255, 255, 255, 0);
    WiFi.config(ip, gateway, subnet);
    WiFi.begin(ssid, password);

    // Wait for connection
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
    }
    server.on("/", handleRoot);
    server.on("/raw", requestData);
    server.on("/cancel", handleCancel);
    server.onNotFound(handleNotFound);
    server.begin();
}

void loop(void) {
    server.handleClient();
}
```

Anexo 12

```
import time
import requests

# CONSTANTES
VERMELHO = '\033[31m'
VERDE = '\033[32m'
BRANCO = '\033[0;0m'
IP = "http://192.168.15.0/raw"

def request_next():
    print("Requisitando de " + VERDE, IP + BRANCO)
    try:
        r = requests.get(IP)
        print('Resposta: ' + r.text)
        return r.text
    except:
        print(VERMELHO + 'Conexão Falhou' + BRANCO)
        return "-2"

def next(response):
    time.sleep(0.002)
    if response < 100:
        return response+1
    else:
        return -1
```

Anexo 13

```
import PySimpleGUI as gui

tb_icon = 'C:/Users/acamp/Desktop/Faculdade/INPE/Projeto/app/img/tb-
icon.png'

def principal():
    return  [
        [gui.Text('Sistema de integração para a coleta de dados da PCD GD
F')],
        [gui.Text('Usuário: ',size=(6,1)), gui.Input(key='usr',size=(20,1
))],
        [gui.Text('Senha: ',size=(6,1)), gui.Input(key='pass',password_ch
ar='•',size=(20,1))],
        [gui.Text('Usuário ou senha inválidos',key='err_usr',text_color='
red',visible=False)],
        [gui.Text('Progresso do Download',key='tex_prog',visible=False)],
        [gui.Text('00000000000000',key='progress',text_color='green',visi
ble=False)],
        [gui.Text('Erro na aquisição!',key='erro',visible=False,text_colo
r='red'),gui.Text('Fim do download!',key='success',visible=False,text_col
or='green')],
        [gui.Button('Download', key='down'),gui.Button('Cancelar',key='ca
ncel',disabled=True),gui.Text(" ",size=(20,1)),gui.Button('Sair',key='sai
r')]
    ]
```

Anexo 14

```
from scrapper import BRANCO, VERMELHO
import time
import PySimpleGUI as gui
import scrapper as scr
import janelas as jan
import utils as u

download = False
response = 0
usuario = 'andrew'
senha = 'pcd'
gui.theme('reddit')
layout = jan.principal()
transferencias = 0
janela = gui.Window('Tela de Login',layout,finalize=True)

while True:

    if download:
        try:
            response = scr.next(transferencias)
            transferencias+=1
        except:
            print(VERMELHO + 'Erro' + BRANCO)

        if int(response) > 0:
            janela['progress'].Update(str(transferencias))

        elif int(response) == -1:
            janela['success'].Update(visible=True)
            download = False

        elif response == -2:
            janela['erro'].Update(visible=True)
            download = False

    eventos, valores = janela.read(timeout=100)
    if eventos == gui.WINDOW_CLOSED or eventos == 'sair':
        print("Sair")
        break
    elif eventos == gui.TIMEOUT_EVENT:
        continue
    elif eventos == 'down':
        print("Download")
        if valores['usr'] == usuario and valores['pass'] == senha:
            janela['cancel'].Update(disabled=False)
            janela['progress'].Update('0',visible=True)
            janela['tex_prog'].Update(visible=True)
            download = True
        else:
            gui.popup_no_titlebar('Erro no login\nUsuário e/ou senha invá
lidos')
    elif eventos == 'cancel':
```

```
            print('Cancelar')
            download = False
            janela['progress'].Update(visible=False)
            transferencias = 0

janela.close()
```