

# Rede LEONA

Rede Colaborativa na América Latina para a Investigação de Eventos Luminosos Transientes e Emissões de Alta Energia de Tempestades

## Primeiros Passos

Detalhamento das etapas para a iniciação dos serviços da Rede LEONA.

## Preparando a Estação LEONA

Descrição das etapas a serem seguidas para garantir o funcionamento das Estações LEONA.

### Hardware

#### Detalhamento das etapas para montagem e configuração dos Hardwares da Estação LEONA

Em desenvolvimento

### Software

#### Detalhamento das etapas para configuração do Software da Estação LEONA

Acessar o servidor de versionamento TortoiseSVN, para isso será necessário realizar o acesso pelo browser.

**Acesse o endereço:**

```
https://10.163.16.7/svn/LEONA/
```

A Estação possui um login único de acesso que tem permissão somente de leitura.

**Login e senha de acesso:**

```
Nome de Usuários: estLEONA  
senha: estLEONA
```

Após o acesso três pastas ficaram disponíveis:

Instalação  
Código Arduino  
LEONA v3 Tornado

## Instalação dos softwares:

A pasta 'Instalação' contém alguns softwares que podem ser necessários na Estação LEONA.

- Os softwares que obrigatoriamente devem ser instalados:

### Controle do arduino:

arduino-1.8.2-windows(x86)

### Drive da placa de aquisição de vídeo:

IVCE-C6XX\_Series\_Driver\_64Bit\_V1.2.5

### Desenvolvimento e uso do software da Estação LEONA:

python-3.6.1

### Versionamento de código ou ultima versão de software:

TortoiseSVN-1.9.7.27907-x64-svn-1.9.7

## Código Arduino

### Requisito:

- IDE Arduino instalado

Esta pasta possui duas outras pastas:

Código osciloscópio

A pasta 'Código osciloscópio' contém o script 'Arduino\_python.ino' que é responsável pela comunicação serial feita no software de teste de taxa de transmissão e gravação de imagens.

Este código deve ser carregado no Arduino conectado a COM3 e COM5.

Pantilt\_001\_28\_08\_15

A pasta 'Pantilt\_001\_28\_08\_15' contém o script 'Pantilt\_001\_28\_08\_15.ino' que é responsável pela comunicação serial feita para o controle de movimento do pantilt.

Este código deve ser carregado no Arduino conectado a COM4.

## **LEONA v3 TORNADO**

Dentro da pasta 'LEONA v3 TORNADO' existe a pasta LEONA TORNADO v3.0 com duas pastas:

documentacao

Contém a documentação detalhada referente ao software .v3.0 da Estação LEONA.

scripts

Contém todos os scripts Python para o funcionamento da Estação LEONA.

Crie uma pasta no desktop da Estação com o nome 'Estação LEONA .v3.0'. Dentro crie duas pastas:

Documentação

Onde deve ser colocados todos os arquivos da pasta 'documentacao' do repositório.

scripts

Onde deve ser colocados todos os arquivos da pasta 'scripts' do repositório.

**Se todos os passos foram corretamente seguidos, a Estação LEONA está pronta para iniciar a transmissão das imagens.**

## **Documentação de Software Estação LEONA .v3.0**

Para ler a documentação e modo de utilização do software de controle e transmissão da Estação LEONA .v3.0 acesse:

### **Documentação de Software**

Esta seção compõe a descrição da arquitetura de software das Estações LEONA. O software das Estações LEONA foi escrito na linguagem Python.

Para iniciar o controle e transmissão da Estação LEONA inicie o script 'main.py' localizado na pasta 'scripts'.

Detalhamento da versão 3.0 do software da Estação LEONA v3.0 - Scripts Python:[🔗](#)  
Cada link desta seção representa um script python.

[settings](#)[🔗](#)

Classe para iniciar as configuração dos Servidores.

---

**class settings.Settings(*n*)**    [\[código fonte\]](#)

Base: `object`

**Parâmetros:** *n* (*inteiro*) – Posição no array de câmeras no arquivo settings.json

**arq\_settings = None**

**Variável arq\_settings():**

Lê Arquivo .json

**camera = None**

**Variável camera(json.array):**

Recebe configuração de cada servidor

**config = None**

**Contante config:**

Lê as configuração de Porta serial de controle de teste de câmeras e do pantilt, a taxa de aquisição de dados e se a versão de teste deve ser ativada

**get\_config()**    [\[código fonte\]](#)

Retorna as configurações lidas do arquivo settings.json

**jsonFile = None**

**Variável jsonFile:**

procura o arquivo .json

**json\_settings = None**

**Variável json\_settings():**

Lê `arq_settings` e transforma em `object.json`

## Arquivo de configuração `settings.json`

Esta seção descreve o arquivo de configuração do software da Estação LEONA.

### **dependencies**

Define as dependências necessárias para que o software possa ser executado.

```
{ "dependencies": {  
    "opencv-python": "3.2.0.7",  
    "tornado": "4.4.2",  
    "pyserial": "3.3",  
    "numpy": "1.12.1"  
}}
```

### **Código descrito acima:**

### **files**

Define uma lista de arquivos que devem existir na pasta do software para que possa ser executado.

### **paths**

Define uma lista de pastas que devem existir na pasta do software para que possa ser executado.

### **Código descrito acima:**

```
{ "files": [  
    "settings.json",  
    "settings.py",  
    "comunicacao_serial.py",  
    "controle_pantilt.py",  
    "servidor_tornado_camera_1.py",  
    "servidor_tornado_camera_2.py",  
    "video_camera.py",  
    "video_camera_teste.py"  
],  
  "paths": [  
    "videos"  
  ]  
}
```

### **cameras**

Define um array de configuração para cada câmera que existe na estação.

### **is\_pantilt**

Define um booleano para indicar a presença de um pan-tilt na estação.

### **controle\_pantilt**

Objeto json que contém as configurações do pan-tilt.

### **comunicacao\_serial**

Objeto json com as configurações de Comunicação Serial do pan-tilt.

### **port\_com**

Define a porta de comunicação.

**baudrate**

Define a taxa de leitura da comunicação.

**video\_camera**

Define um objeto de configuração da classe videoCamera.

**frames**

Define a taxa de aquisição das câmeras.

**port\_listen**

Define a porta do websocket do servidor tornado.

**is\_test**

Define um booleano para indicar se deverá iniciar em modo de teste.

**teste**

Define um objeto contendo os dados para teste do software.

**video\_camera**

Define um objeto de configuração da classe videoCamera.

**port\_com**

Define a porta de comunicação com o arduino.

**baudrate**

Define a taxa de leitura da comunicação com o arduino.

**name\_arq**

Define o nome do arquivo onde será escrito os tempos de cada aquisição da classe videoCamera.

**Código descrito acima:**

```
{
  "cameras": [
    {
      "is_pantilt": "False",
      "controle_pantilt": {
        "comunicacao_serial": {
          "port_com": "COM4",
          "baudrate": "9600"
        }
      },
      "video_camera": {},
      "frames": 30,
      "port_listen": "8888",
      "is_test": "False",
      "test": {
        "video_camera": {
          "port_com": "COM5",
          "baudrate": "9600",
          "name_arq": "Camera1.txt"
        }
      }
    }
  ]
}
```

**main**

Inicia todos os serviços da Estação LEONA.

---

**main.camera**(*arg*)    [\[código fonte\]](#)

Recebe comando para iniciar servidores

**Parâmetros:** *arg* (string) – Comando de inicialização da câmera 1 ou 2

---

`main.main()` [\[código fonte\]](#)

Verifica se todos os arquivos e dependências estão presentes na Estação LEONA e se estiver inicia o serviço de transmissão.

---

`main.verificar_arquivos(files)` [\[código fonte\]](#)

Verifica se todos os arquivos necessários para execução do software existem

**Parâmetros:** `files` (*json.array*) – Lista json contendo os arquivos que deverão existir na pasta do software

---

`main.verificar_dependencias(dependencies)` [\[código fonte\]](#)

Verifica se todos as dependências necessárias para execução do software existem

**Parâmetros:** `dependencies` (*json.array*) – Lista json contendo as dependências que deverão existir para execução do software

---

`main.verificar_pastas(paths)` [\[código fonte\]](#)

Verifica se todos as pastas necessárias para execução do software existem

**Parâmetros:** `paths` (*json.array*) – Lista json contendo as pastas que deverão existir na pasta do software

`servidor_tornado_camera_1`

Classe responsável pela transmissão das imagens.

---

`class servidor_tornado_camera_1.ApiHandler(application, request, **kwargs)` [\[código fonte\]](#)

Base: `tornado.web.RequestHandler`

`get(*args)` [\[código fonte\]](#)

`post()` [\[código fonte\]](#)

---

`class servidor_tornado_camera_1.SocketHandler(application, request, **kwargs)` [\[código fonte\]](#)

Base: `tornado.websocket.WebSocketHandler`

`check_origin(origin)` [\[código fonte\]](#)

`on_close()` [\[código fonte\]](#)

`open()` [\[código fonte\]](#)

Inicia o envio de dados

`render()` [\[código fonte\]](#)

Prepara imagens para transmissão

`writer= None`

---

`servidor_tornado_camera_1.config`

```
= {'is_pantilt': 'False', 'port_con': 'COM4', 'baudrate': '9600', 'frames': 30, 'port_listen': '8888', 'is_test': 'False', 'vc_port_con': 'COM5', 'vc_baudrate': '9600', 'vc_name_arq': 'Camera1.txt'}
```

Variável config (settings.Settings):

Lê o arquivo de configuração. Setting(0). Não alterar pois é o que determina qual camera será inicialização

---

`servidor_tornado_camera_1.queue= <multiprocessing.queues.Queue object>`

Variável queue: inicia a pilha para armazenar imagens. Paramêtro: maxsize(330)–> Controla o buffer máximo de leitura da câmeras, NÃO ALTERAR!, esta variável, junto a de “frames” presente no arquivo de configuração, garante que a sincronia do serviço.

---

`servidor_tornado_camera_1.vc= <VideoCamera(Thread-1, started daemon 5452)>`

Variável vc: (instância) Inicializa leitura das câmeras

---

`servidor_tornado_camera_2`

---

`class servidor_tornado_camera_2.ApiHandler(application, request, **kwargs)` [\[código fonte\]](#)

Base: tornado.web.RequestHandler

`get(*args)` [\[código fonte\]](#)

`post()` [\[código fonte\]](#)

---

`class servidor_tornado_camera_2.SocketHandler(application, request, **kwargs)` [\[código fonte\]](#)

Base: tornado.websocket.WebSocketHandler

`check_origin(origin)` [\[código fonte\]](#)

`on_close()` [\[código fonte\]](#)

`open()` [\[código fonte\]](#)

Inicia o envio de dados



`render()` [\[código fonte\]](#)

Prepara imagens para transmissão

`writer = None`

---

`servidor_tornado_camera_2.config`

```
= {'is_pantilt': 'False', 'port_con': ' ', 'baudrate': ' ', 'frames': 30, 'port_listen': '8889', 'is_test': 'False',  
'vc_port_con': 'COM3', 'vc_baudrate': '9600', 'vc_name_arq': 'Camera2.txt'}
```

Variável config (settings.Settings):

Lê o arquivo de configuração. Setting(1). Não alterar pois é o que determina qual camera será inicialização

---

`servidor_tornado_camera_2.queue = <multiprocessing.queues.Queue object>`

Variável queue: inicia a pilha para armazenar imagens. Parâmetro: maxsize(330) -> Controla o buffer máximo de leitura da câmeras, NÃO ALTERAR!, esta variável, junto a de "frames" presente no arquivo de configuração, garante que a sincronia do serviço.

---

`servidor_tornado_camera_2.vc = <VideoCamera(Thread-2, started daemon 1096)>`

Variável vc: (instância) Inicializa leitura das câmeras

---

video\_camera

---

`class video_camera.VideoCamera(path=0, queue=None)` [\[código fonte\]](#)

Base: `threading.Thread`

Classe que estende `threading.Thread` responsável por ler uma imagem da placa de captura colocando a mesma em uma `multiprocessing.Queue`

Inicia a câmera

- Parâmetros:
- **path** (*inteiro*) – Indica qual placa de captura deve ser utilizada. Número identido ao indicado na placa conectada a placa mãe da Estação Leona.
  - **queue** (*multiprocessing.Queue*) – array responsável por guardar os frames para que outro processo possa ler

`out = None`

Variável out: (`cv2.VideoWriter`) Instância de `VideoWriter`

`queue = None`

Variável queue: (`multiprocessing.Queue`) Instância de `Queue`

**recorded = None**

**Variável recorded:**

(boolean) Controla inicialização e finalização da gravação

**run()** [\[código fonte\]](#)

Método executado pela Thread

**start\_record()** [\[código fonte\]](#)

Inicia a gravação em disco das imagens capturadas pela câmera

**stop()** [\[código fonte\]](#)

Finaliza a leitura da camera

**stop\_record()** [\[código fonte\]](#)

Finaliza a gravação em disco das imagens capturadas pela câmera

**stopped = None**

**Variável stopped:**

(boolean) Controla finalização da aquisição de imagem

**stream = None**

**Variável stream:**

(cv2.VideoCapture) Instância de VideoCapture

video\_camera\_teste

## Classe de Teste de gravação de frames e transmissão dos dados

Estende `threading.Thread` responsável por ler uma imagem da placa de captura colocando a mesma em uma `multiprocessing.Queue`

Debug

Osciloscópio

Permite o monitoramento do loop com osciloscópio para o mesmo é necessário que no arquivo de configuração a variável `TEST_OSCILOSCOPIO` esteja como `True` e que passe como parametro um objeto da classe `comunicacao.serial`

Arquivo txt

Gera um arquivo txt contendo o timestamp do instante de cada loop

---

**`class video_camera_teste.VideoCamera(path=0, queue=None, com=None, arquivo=None)`**  
[\[código fonte\]](#)

Base: `threading.Thread`

Inicia a câmera

- Parâmetros:**
- **`path`** (*String*) – recebe caminho de leitura da câmera
  - **`queue`** (*`multiprocessing.Queue`*) – array responsável por guardar os frames para que outro processo possa ler
  - **`com`** (*`comunicacao.serial.ComunicacaoSerial`*) – recebe a com responsável pelo arduino que recebera o comando para DEBUG
  - **`arquivo`** (*String*) – Nome do arquivo gerado para DEBUG dos tempos de aquisição

**`arquivo = None`**

**Variável arquivo:**

(String) Caminho e/ou nome do arquivo

**`com = None`**

**Variável com:** (`ComunicacaoSerial`) Instância de `ComunicacaoSerial`

**`out = None`**

**Variável out:** (`cv2.VideoWriter`) Instância de `VideoWriter`

**queue = None**

Variável queue: (multiprocessing.Queue) Instância de Queue

**recorded = None**

Variável recorded:

(boolean) Controla inicialização e finalização da gravação

**run()** [\[código fonte\]](#)

Método executado pela Thread

**start\_record()** [\[código fonte\]](#)

Inicia a gravação em disco das imagens capturadas pela câmera

**stop()** [\[código fonte\]](#)

Finaliza a leitura da camera

**stop\_record()** [\[código fonte\]](#)

Finaliza a gravação em disco das imagens capturadas pela câmera

**stopped = None**


Variável stopped:

(boolean) Controla finalização da aquisição de imagem

**stream = None**

Variável stream (cv2.VideoCapture):

Instância de VideoCapture

comunicacao\_serial 

Classe responsável por criar um canal de comunicação com o arduino.

---

```
class comunicacao_serial.ComunicacaoSerial(port='COM3', baudrate=9600) \[código fonte\]
```

Base: `object`

Inicia a Comunicação Serial

- Parâmetros:**
- **port** (*string*) – Recebe a porta COM em que o Arduino foi conectado, por padrão, porta COM3
  - **baudrate** (*inteiro*) – Taxa de aquisição de dados feita pelo Arduino, por padrão 9600

**baudrate = None**

**Variável baudrate:**

Recebe a taxa de aquisição de dados feita pelo Arduino

**enviar(arg)** [\[código fonte\]](#)

Recebe o dado a ser enviado para o Arduino e depois limpa a memória para permitir novo envio de dados

**Parâmetros:** **arg** (*inteiro*) – Dado para teste da taxa de leitura das cameras. Teste feito com Osciloscópio e Arduino: taxa deve ser 30ms.

**ser = None**

**Função ser:** Abre comunicação a Comunicação Serial e é responsável por ler port e baudrate definidos

controle\_pantilt

Classe responsável por criar uma interface de comunicação com o arduino e o Pan-Tilt.

---

**class controle\_pantilt.ControlePantilt(con=None)** [\[código fonte\]](#)

Base: `object`

Iniciar a posição do Pan-Tilt

**Parâmetros:** **con** (*object*) – Instância de comunicacao serial

Padrão de comando de comunicação entre Arduino e Pan-Tilt:

```
!000U* - Mover para cima
!000D* - Mover para baixo
!000R* - Mover para direita
!000L* - Mover para esquerda
!1110* - Ligar camera (um,um,um,zero)
!111F* - Desligar camera
```

**Valores máximos e mínimos:**

Azimuth: 0° -> 350° Elevação: -35° -> +35°

**azGraus = None**

**Variável azGraus:**

(inteiro) Armazena a ultima posicao do pan-tilt em azimuth

#### **azimute(*graus*)** [\[código fonte\]](#)

Verifica se os graus recebidos estão entre 0 e 350, compara com o valor anterior e assim determina se o comando enviado deve ser para a direita ou esquerda

**Parâmetros:** *graus* (*inteiro*) – valor entre 0 e 350 azimuth

#### **con = None**

**Variável con:** (instância) Recebe a comunicação serial iniciado pela classe principal

#### **converterelevacao(*graus*)** [\[código fonte\]](#)

Converte o valor recebido para o valor esperado pelo controlador de Pan-Tilt, onde  $-35^{\circ} == 0$ ,  $0^{\circ} == 35$ ,  $35^{\circ} == 70$ , ou seja todos os valores recebidos devem ser acrescidos de 35

**Parâmetros:** *graus* (*inteiro*) – valor entre -35 e 35 elevação

#### **descer(*graus*)** [\[código fonte\]](#)

Concatena os graus com !###D\* onde #### representa os graus, e envia para a controladora

**Parâmetros:** *graus* (*inteiro*) – valor a ser enviado para controlador

#### **desligarcamera()** [\[código fonte\]](#)

Desliga as câmeras: Comando deve ser sempre executado após as observações agendadas, ou horários determinado na Estação LEONA, para evitar queimar as câmeras CCDs.

#### **direita(*graus*)** [\[código fonte\]](#)

Concatena os graus com !###R\* onde #### representa os graus, e envia para a controladora

**Parâmetros:** *graus* (*inteiro*) – valor a ser enviado para controlador

#### **elGraus = None**

**Variável elGraus:**

(inteiro) Armazena a ultima posicao do pan-tilt em elevacao

#### **elevacao(*graus*)** [\[código fonte\]](#)

Verifica se os graus recebidos estão entre 0 e 70, compara com o valor anterior e assim determina se o comando enviado deve ser para cima ou baixo

**Parâmetros:** `graus` (*inteiro*) – valor entre -35 e 35 elevação

**esquerda(*graus*)** [\[código fonte\]](#)

Concatena os graus com `!###L*` onde `####` representa os graus, e envia para a controladora

**Parâmetros:** `graus` (*inteiro*) – valor a ser enviado para controlador

**formatargraus(*graus*)** [\[código fonte\]](#)

Verifica valor informado e formata no padrão de bits

**Parâmetros:** `graus` (*inteiro*) – valor de azimuth ou elevação selecionado pelo observador da Rede Leona

**ligarcamera()** [\[código fonte\]](#)

Liga as câmeras:

**resetarazimute()** [\[código fonte\]](#)

Envia para a controladora `!350L`, assim resetando azimuth em 0°

**resetarelevacao()** [\[código fonte\]](#)

Envia para a controladora `!070D`, assim resetando a elevação em 0° ou -35°

**subir(*graus*)** [\[código fonte\]](#)

Concatena os graus com `!###U*` onde `####` representa os graus, e envia para a controladora

**Parâmetros:** `graus` (*inteiro*) – valor a ser enviado para controlador

## Gerar automaticamente a documentação de software¶

A documentação da Estação LEONA v3.0 foi gerada automaticamente com o uso de biblioteca Sphinx, para utilizar o mesmo método com esta ou outras versões do software em python basta seguir os passos abaixo.

## Requisitos¶

Para utilizar a biblioteca Sphinx é necessário que a versão 3.6.1 ou superior do python esteja instalada. Para instalar você pode checar a seção [Software - Detalhamento das etapas para configuração do Software da Estação LEONA](#)

## Preparando o esquema básico de documentação¶

Instalar:

```
pip install sphinx
pip install sphinx_rtd_theme
```

Na pasta dos scripts execute o comando:

```
md documentacao scripts rst
```

Para iniciar a biblioteca Sphinx execute o comando:

```
sphinx-quickstart
```

Na tela do quickstart altere os comandos:

```
Project Name: [coloque o nome]
Author name: [coloque o nome]
Project Version: [coloque a versão]
Project release: [coloque a versão de produção]
Project language: pt_BR
autodoc[y]: y
viewcode[y]: y
Create Makefile[y]: y
Create Windows command file[y]: y
```

No arquivo conf.py modifique as linhas:

- Substitua as linhas:

```
# import os
# import sys
# sys.path.insert(0, os.path.abspath('.'))
```

- Pelas linhas:

```
import os
import sys
sys.path.insert(0, os.path.abspath('../scripts'))
import sphinx_rtd_theme
```

- Para garantir que a a def \_\_init\_\_ será automaticamente documentada adicione a linha:

```
autoclass_content = 'both'
```

- Na seção # – General configuration –



Na linha:

```
# The master toctree document
master_doc = 'index'
```

Altere 'index' para:

```
master_doc = 'RedeLeonaDoc'
```

- Na linha # – Options for HTML output --#

Comente:

```
html_theme = 'alabaster'
```

E insira:

```
html_theme = "sphinx_rtd_theme"
html_theme_path = [sphinx_rtd_theme.get_html_theme_path()]
```

Execute o comando:

```
make html
```

Aviso: Todos os scripts python devem estar na pasta "scripts"! Executar o comando:

```
sphinx-apidoc -o ./rst ./scripts
cd rst
md _static
```

Recorte o script "conf.py" e "index.rst" para a pasta criada "rst".

Execute o comando:

```
cd ..
```

Execute o comando:

```
sphinx-build -b html ./rst ./documentacao
```

Pronto! A estrutura básica de documentação foi gerada!

Você pode conferir na pasta “documentacao” na arquivo ‘RedeLeonaDoc.html’

Exemplo de código python para gerar auto documentação¶

Código de Exemplo do padrão de documentação em Python:

```
import cv2, time

class testeCamera:
    """Classe para testar funcionamento da câmera"""

    def __init__(self, captura):
        """Recebe camera

        :type captura: object
        :param captura: recebe caminho de leitura da câmera
        """

        self.captura0 = cv2.VideoCapture(0)
        #time.sleep(3)
        #captura1 = cv2.VideoCapture(0)

    def start(self):
        """ Inicia leitura da camera"""

        while(1):
            ret0, frame0 = self.captura0.read()
            """variável frame0: (matrix de string) Imagens capturada da câmera"""
            cv2.imshow("Video - 0", frame0)

            k = cv2.waitKey(30) & 0xff
            if k == 27:
                break

    def __del__(self):
        """Finaliza transmissão da camera"""

        captura0.release()
        #captura1.release()
        cv2.destroyAllWindows()
```

**AVISO:** O arquivo settings.json não é automaticamente documentado, qualquer alteração deve ser feita diretamente no arquivo ‘settings.rst’, seção: ‘Arquivo de configuração settings.json’.

Referências¶

- Tutorial em vídeo:

[Video Youtube em inglês.](#)

- Exemplo de documentação:

[Pythonhosted.](#)

[Sphinx.](#)

## Check List de Configurações

Em desenvolvimento

# Busca

- Índice
- Índice do Módulo
- [Página de Busca](#)