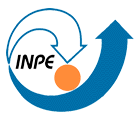
****

**INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS**

**DIVISÃO DE GEOFÍSICA ESPACIAL**

**DIVISÃO DE DESENVOLVIMENTO DE SISTEMAS DE SOLO**

**ENGENHARIA E TECNOLOGIA ESPACIAL**

**Síntese de abordagens adotadas na revisão do Projeto LEONA: Servidor, transmissão e aquisição de imagens**

2017

**Sumário**

[1 - INTRODUÇÃO 3](#__RefHeading___Toc910_742393069)

[2 - Revisão e Redefinição de tecnologias aplicadas à rede leona 4](#__RefHeading___Toc912_742393069)

[3 - Servidor nas estações leona 5](#__RefHeading___Toc920_742393069)

[3.1 - FLASK 5](#__RefHeading___Toc926_742393069)

[3.1.1 - UDP (*User Datagram Protocol*) 5](#__RefHeading___Toc932_742393069)

[3.1.2 - PROTOCOLO TCP *(Transmission Control Protocol )* 5](#__RefHeading___Toc940_742393069)

[3.1.3 - Combinações de Flask e UDP e TCP 6](#__RefHeading___Toc948_742393069)

[3.2 - TORNADO 6](#__RefHeading___Toc956_742393069)

[3.3 - Tornado e Flask 7](#__RefHeading___Toc962_742393069)

[4 - PLACA DE AQUISIÇÃO DE VÍDEO 7](#__RefHeading___Toc968_742393069)

[5 - SUBPROCESSO E MULTIPROCESSAMENTO EM PYTHON 7](#__RefHeading___Toc500_370054860)

[5.1 - Subprocessos e Multiprocessamento 8](#__RefHeading___Toc980_7423930691)

[5.2 - Sincronismo 8](#__RefHeading___Toc990_7423930691)

# INTRODUÇÃO

Este documento tem por objetivo apresentar as abordagens adotadas na revisão e redefinição de tecnologias aplicadas até maio de 2017, que aqui será nomeada de ‘.v0’, no Projeto LEONA, referentes à: captura, gravação de vídeo, transmissão ao vivo das imagens adquiridas nas estações e construção da Rede LEONA.

Este documento não possui detalhamento referente à arquitetura e funcionalidades do software da Estação LEONA, entretanto, a documentação, em desenvolvimento, está presente no arquivo ‘RedeLeonaDoc.html’ junto ao código do projeto.

# Revisão e Redefinição de tecnologias aplicadas à rede leona

No intuito de iniciar uma revisão na ‘.v0’ do projeto de software da Rede LEONA, foi realizado um estudo para elencar possíveis problemas na versão de produção do software. Os seguintes problemas foram observados:

1. Criação de uma nova estação causa uma falha no sistema operacional Windows 7;
2. Uso indevido do datador de tempo em vídeo IOTA;
3. Transmissão e acesso a somente uma câmera;
4. Requisito de captura de trinta imagens por segundo não é atendido.

Após estudo na literatura e revisão junto ao fornecedor da placa de aquisição, foram constatados que os problemas 1 e 2 eram respectivamente provenientes do uso de uma versão incompatível de software para a placa de aquisição de vídeo e não cumprimento do manual do fabricante/fornecedor. Com a instalação da versão indicada pelo fornecedor, versão 1.2.5 de 64 bits, o problema 1 foi solucionado e o problema 2 é automaticamente corrigido quando o IOTA sincroniza com mais de 3 satélites, assim como descrito no manual.

Para os problemas 3 e 4, um plano de refatoração de código foi inicializado, porém a linguagem JAVA, utilizada na ‘.v0’, mostrou-se ineficiente para a aplicação LEONA, uma vez que mesmo com o uso de artifícios da linguagem, como *threads* (programa em execução com múltiplos fluxos de execução), não foi possível atingir a taxa de gravação esperada, trinta imagens por segundo. Assim, após breve estudo na literatura a linguagem Python foi escolhida para compor a nova bateria de testes no software LEONA.

Foram realizados dois testes com a linguagem: análise da taxa de leitura da câmera pelo software; e taxa de gravação em disco das imagens adquiridas. Ambos deveriam atingir uma taxa de aquisição de trinta imagens por segundo. Para isso, o OPENcv (*Open Source Computer Vision Library*), uma biblioteca para manipulação de imagens, foi utilizada junto ao Python para tratar da aquisição das imagens e um osciloscópio realizou a medição da taxa de aquisição e gravação destas imagens. Os resultados comprovaram que a linguagem Python atende os requisitos estipulados nos testes, soluciona o problema número 4 e que a linguagem pode, então, ser utilizada para o novo software da Rede LEONA que substituirá o que atualmente se encontra nas estações. O Problema número 3 será abordado no capítulo 5 deste documento.

# Servidor nas estações leona

Uma das principais funcionalidades da Rede LEONA é fornecer um serviço de transmissão de imagens ao vivo, ao qual, se dará por meio de um portal em que o observador/pesquisador poderá ter acesso online. Neste intuito, foi inicializado a construção do servidor LEONA com a linguagem Python.

## FLASK

Flask é um *microframework web* que permite criar um servidor e é facilmente utilizado como uma biblioteca do Python. Flask foi utilizado para renderizar as imagens capturadas pela câmera, permitindo sua transmissão pela web.

Neste título entendesse o termo “transmissão das câmeras”, ou similar, até o subtitulo 3.1.3 como: transmissão da imagem capturada de somente uma câmera que é duplicada.

### **UDP (***User Datagram Protocol***)**

O UDP (em português: Protocolo de Datagrama de Usuário) é um protocolo de comunicação que se baseia no envio de pacotes de informações, é muito conhecido na literatura para realizar transmissão ao vivo e *broadcast (*transmissão para múltiplos clientes*)* de dados. O protocolo não possui garantia de verificação erros ou ordem de envio dos dados.

No protoclo UDP o tamanho máximo de transmissão de dados é de 65507 bytes, ou o equivalente a uma imagem de tamanho 320x240, muito inferior ao padrão das câmeras utilizadas na Rede LEONA, 720x480. Para garantir a transmissão ao vivo das câmeras, as imagens adquirias foram redimensionadas em um pacote menor, metade de seu tamanho, enviadas, e redimensionadas.

Foram realizados testes para verificar a experiência dos usuários durante a transmissão ao vivo. No ambiente de teste foi possível realizar a transmissão das câmeras com acesso simultâneo de setenta usuários no servidor e realizar a gravação de uma câmera com uso de *threads*, porém, devido ao redimensionamento das imagens a qualidade da transmissão ao vivo recebida pelo observador seria prejudicada, o que inviabilizou o uso do protocolo UDP.

### **PROTOCOLO TCP *(Transmission Control Protocol )***

Assim como o UDP, o protocolo TCP (em português: Protocolo de Controle de Transmissão) também é um protocolo de comunicação, o que o diferencia é a capacidade de controlar o tamanho dos pacotes enviados e verificar se o que foi recebido é igual ao que foi enviado. Por causa destas características o protocolo foi escolhido para substituir o UDP.

No protocolo TCP, cada nova conexão representa um novo caminho de conexão entre cliente, que recebe o dado, e servidor, que envia o dado, aberto (*socket*), o que resulta em novo processo, em nossa aplicação uma *thread,* gerando consumo gradativo de CPU.

Partindo do requisito de um número máximo de quinze observadores, e priorizando a taxa de gravação de trinta imagens por segundo, testes foram realizados para analisar o consumo de CPU durante o cumprimento destes dois requisitos. Mesmo com a otimização do código, as máquinas atuais das Estações LEONA, as mesmas utilizadas durante os testes, não suportariam mais do que três conexões simultâneas sem perder imagens durante a gravação, requisito de maior prioridade. Não atendendo os requisitos dos testes, o protocolo foi inviabilizado para esta aplicação.

### Combinações de Flask e UDP e TCP

Vale ressaltar, a fim de disseminar conhecimento, testes de resultado positivo e negativo durante este desenvolvimento e algumas combinações e tentativas feitas com os protocolos e soluções apresentados neste capítulo.

* Uso de dois Flasks, um como cliente e um como servidor, em máquinas diferentes para distribuir o consumo de CPU: aumento de três para seis usuários com TCP; não resolve, em UDP ou TCP, perda de imagens na gravação;
* Cliente e servidor rodando em máquinas diferentes, não ajudou de forma significativa na redução do consumo de CPU.

## TORNADO

Tornado é um *web framework* assíncrono que, de forma otimizada, trata as conexões recebidas usando protocolos de comunicação como TCP, Websocket.

Neste capítulo entendesse o termo “transmissão das câmeras”, ou similar, até o subtitulo 3.2.1, como: transmissão da imagem de duas câmeras, cada uma conectada a uma placa de vídeo. Abordagem adotada para simular o envio de imagens de duas câmeras, uma com IOTA e outra sem, conforme arquitetura proposta por *stakeholder.*

## Tornado e Flask

Nesta aplicação o Tornado foi utilizado para disponibilizar em uma porta do endereço IP do servidor LEONA a transmissão ao vivo das câmeras. Este procedimento foi utilizado para aproveitar a capacidade da solução Tornado em otimizar a transmissão de dados com métodos assíncronos e permitir que o Flask ficasse responsável somente por receber os comandos de movimento das câmeras, gravação das imagens e acesso de observadores a transmissão.

Com esta arquitetura de software foi possível atender os requisitos de gravação, trinta imagens por segundo, e o acesso de quinze observadores simultâneos. Esta versão serviu de base para a arquitetura de software adotada nesta nova versão da Rede LEONA (.v1), faltando a resolução do problema 3

# PLACA DE AQUISIÇÃO DE VÍDEO

No intuito de resolver o problema 3 descrito neste documento, Iniciou-se um estudo da *Application Programming Interface (*API), em tradução para o português: interface de programação de aplicativos, disponibilizada pelo fabricante da placa de aquisição de imagens. O requisito era obter imagens de duas câmeras utilizando a mesma placa.

A API é escrita na linguagem C/C++ e necessita de uma *Dynamic-link library* (biblioteca de vínculo dinâmico) ou mais conhecida por DLL, que possui funções como a escolha de câmeras para aquisição de imagens.

Em Python é possível fazer a leitura de funções escritas na linguagem C/C++, mas para isso o código ou a DLL deve ter sido escrita, por exemplo, em Cython, que é uma linguagem de programação para simplificar a escrita de módulos de extensão para Python, ou soluções similares, o que permite que a linguagem Python possa interpretar estas funções/métodos ou classes importadas, assim, uma vez que a DLL disponibilizada junto a placa de aquisição de imagens é um código fechado e criptografado, não foi possível modificar o código interno para o padrão necessário, sendo necessário buscar soluções em Python para a leitura de duas câmeras usando a placa disponível no projeto.

# **SUBPROCESSO E MULTIPROCESSAMENTO EM PYTHO**

Após uma sequência de testes, foi identificado que para a placa de aquisição de imagens fazer a leitura de mais de uma câmera é preciso iniciar um processo separado da sequência principal do código de leitura das câmeras, o que resolve o problema 3 descrito neste documento. Em Python pode-se iniciar um processo de diferentes maneiras, como, por exemplo: *threads*, subprocesso e multiprocessamento.

## Subprocessos e Multiprocessamento

Subprocessos em Python permitem que um *script* seja iniciado sem pertencer ao mesmo espaço de memória do *script* que o inicializou. Já o multiprocessamento permite que de modo análogo as threads os pacotes ou bibliotecas inclusas no script Python possam ocorrer localmente ou de modo remoto, ambos artifícios pertencem a biblioteca padrão do Python.

O subprocesso foi utilizado junto ao servidor Tornado e Flask para inicializar a leitura das câmeras, apesar de realizar a leitura das duas câmeras, atendendo a arquitetura proposta da Rede LEONA, ao iniciar um subprocesso não é possível enviar ou receber dados entre os *scripts*, pois cada um é iniciado como um programa diferente.

No multiprocessamento é possível estabelecer uma via de comunicação entre *scripts* diferentes por meio do uso de *Queues,* modulo que pertencente a biblioteca padrão do Python que permite a troca de dados de forma segura e sem perdas, entre diferentes processos. Com o use deste artifício, a ‘.v2’ da Rede LEONA foi escrita, agora atendendo a dois requisitos fundamentais: Transmissão de duas câmeras e acesso de quinze observadores na estação.

Testes mais detalhados, para garantir a taxa de aquisição de trinta imagens por segundo, demostraram que esta combinação de artifícios causa uma grande variação no consumo de CPU e memória RAM, que as atuais máquinas disponíveis nas Estações LEONA, não são capazes de estabilizar, o que faz com que esta solução não atenda o requisito na gravação das imagens em disco, pois perde quadros.

## Sincronismo

Para resolver o problema de perda de imagens durante a gravação é necessário permitir que a leitura e escrita de dados trabalhem de forma síncrona, criando um meio de comunicação entre os scripts Python de gravação e transmissão de imagens. A solução adotada, que descreve a ‘.v3’ da Rede LEONA, foi tratar e controlar ambos pelo script Python responsável pela leitura das câmeras e que fornece as imagens ao servidor Tornado. Esta troca de dados é feita com uso de uma *queue* que permite o controle do tamanho máximo de dados que é disponibilizado ao servidor Tornado por segundos, que por sua vez também acessa esse dado em uma determinada frequência, trinta vezes por segundo, seguindo o padrão de gravação descrito no requisito da Estação LEONA.

Para otimizar o consumo de CPU e memória RAM, as funções que antes eram controladas pelo servidor Flask, na ‘.v2’ da Rede LEONA foram reescritas no servidor Tornado, permitindo que qualquer outra solução possa ser adotada no Servidor Central da Rede LEONA.

# **DETALHAMENTO DA ‘.V3.0’ DO SOFTWARE DAS ESTAÇÕES LEONA**

# primeiros passos

* + 1. **Preparando a Estação LEONA**

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

* + - 1. **Hardware**

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

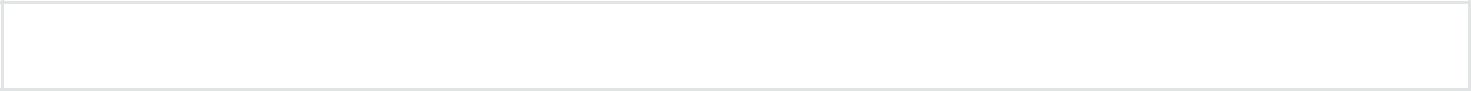
Em desenvolvimento

* + - 1. **Software**

Detalhamento das etapas para configuração do Software da Estação LEONA

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

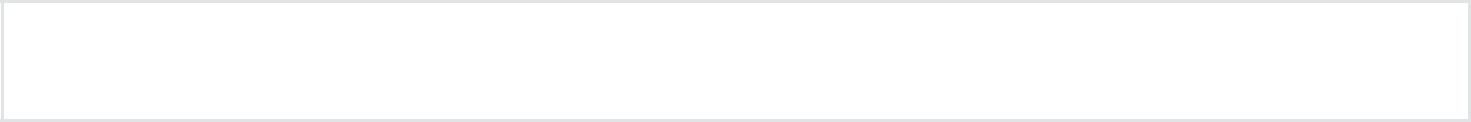
Acesse o endereço:



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 disponiveis:

Instalação

Código Arduino

LEONA v3 Tornado

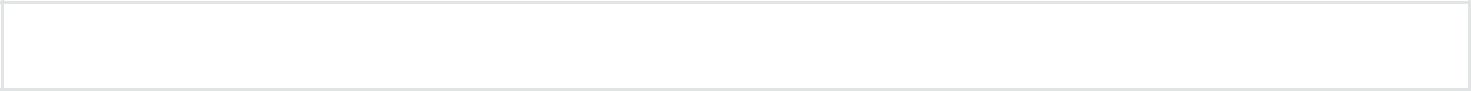
Instalação dos softwares:

A pasta ‘Instalação’ contém alguns so wares que podem ser necessários na Estação LEONA.

Os so wares que obrigatóriamente devem ser instalados:

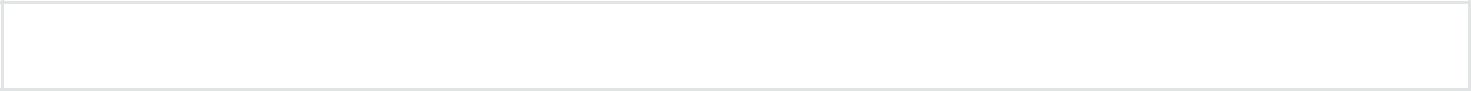


Controle do arduino:



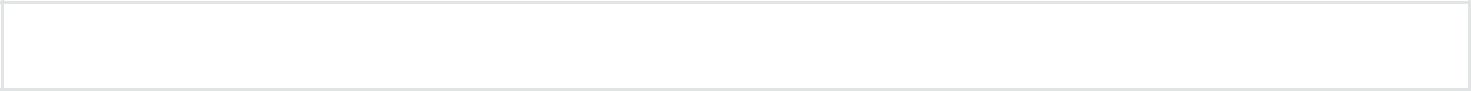
arduino-1.8.2-windows(x86)

Drive da placa de aquisição de vídeo:



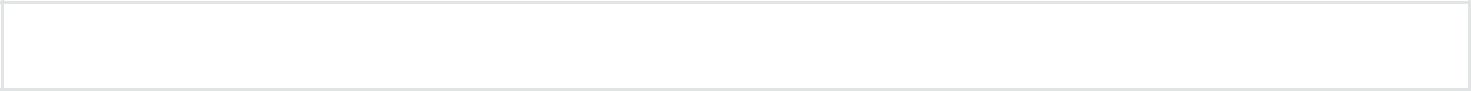
IVCE-C6XX\_Series\_Driver\_64Bit\_V1.2.5

Desenvolvimento e uso do so ware da Estação LEONA:



python-3.6.1

Versionamento de código ou ul ma versão de so ware:



TortoiseSVN-1.9.7.27907-x64-svn-1.9.7

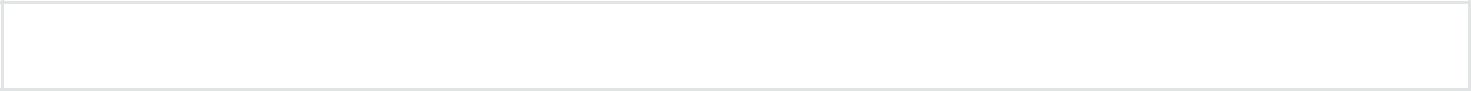
Código Arduino

Requisito:

IDE Arduino instalado



Esta pasta possui duas outras pastas:



Código osciloscopio

A pasta ‘Código osciloscopio’ contém o script ‘Arduino\_python.ino’ que é responsável pela comunicação serial feita no so ware de teste de taxa de transmissão e gravação de imagens.

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

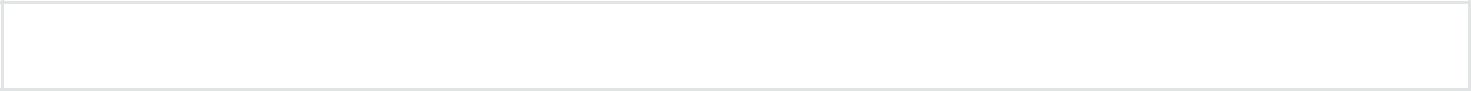
Pantilt\_001\_28\_08\_15

A pasta ‘Pan lt\_001\_28\_08\_15’ contém o script ‘Pan lt\_001\_28\_08\_15.ino’ que é responsável pela comunicação serial feita para o controle de movimento do pan lt.

Este código deve ser carregado no Arduino conectado a COM4.

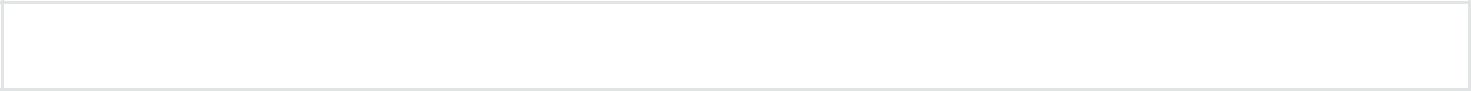
LEONA v3 TORNADO

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



documentacao

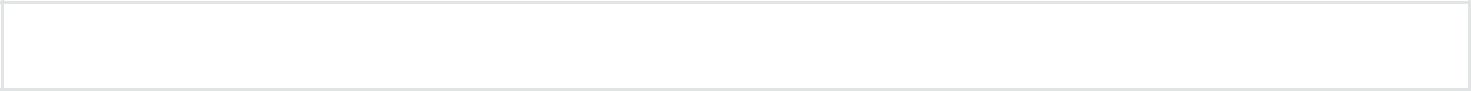
Contém a documentação detalhada referente ao so ware .v3.0 da Estação LEONA.



scripts

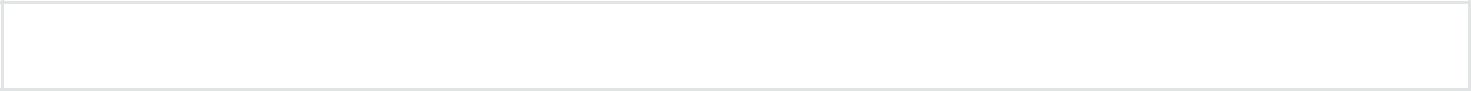
Contém todos os scripts Python para o funcionamento da Estação LEONA.

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 u lização do so ware de controle e transmissão da

Estação LEONA .v3.0 acesse:

Documentação de Software

Esta seção compôe a descrição da arquitetura de so ware das Estações LEONA. O so ware 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 so ware da Estação LEONA v3.0 - Scripts Python:[¶](#page4) Cada link desta seção representa um script python.

se ngs[¶](#page4)

Classe para iniciar as configuração dos Servidores.



class settings.Settings(n) [[código fonte]](file:///C:/Users/LEONA/Documents/Projetos Python/Projeto Leona/LEONA/LEONA v3 Tornado/LEONA TORNADO v3.0/documentacao/_modules/settings.html" \l "Settings)

Base: object



Parâmetros: n (inteiro) – Posição no array de câmeras no arquivo se ngs.json



arq\_settings= None

Variável arq\_se ngs():

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 pan lt, a taxa de aquisição de dados e se a versão de teste deve ser a vada



get\_config() [[código fonte]](file:///C:/Users/LEONA/Documents/Projetos Python/Projeto Leona/LEONA/LEONA v3 Tornado/LEONA TORNADO v3.0/documentacao/_modules/settings.html" \l "Settings.get_config)

Retorna as configurações lidas do arquvo se ngs.json



jsonFile= None

Variável jsonFile:

procura o arquivo .json



json\_settings= None

Variável json\_se ngs():

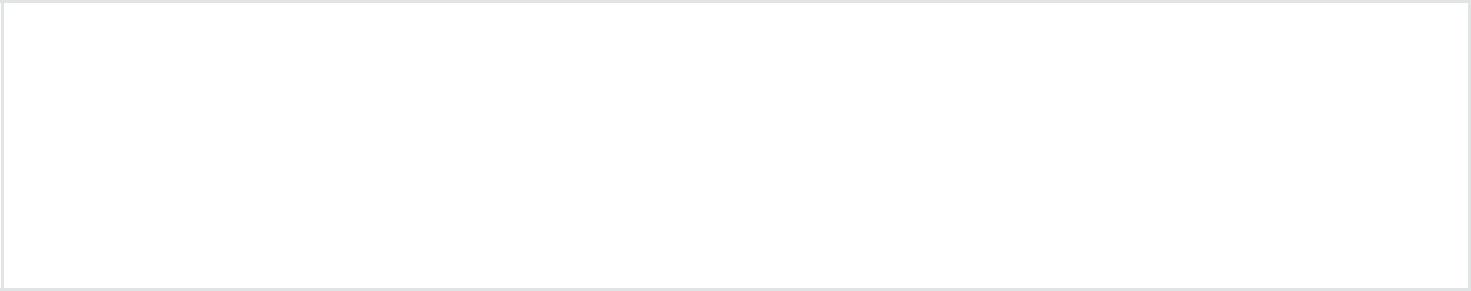
Lê arq\_se ngs e transforma em object.json

Arquivo de configuração se ngs.json[¶](#page5)

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

dependencies

Define as dependências necessárias para que o so ware possa ser executado.



{"dependencies":{

"opencv-python":"3.2.0.7",

"tornado":"4.4.2",

"pyserial":"3.3",

"numpy":"1.12.1"

}}

Código descrito acima:

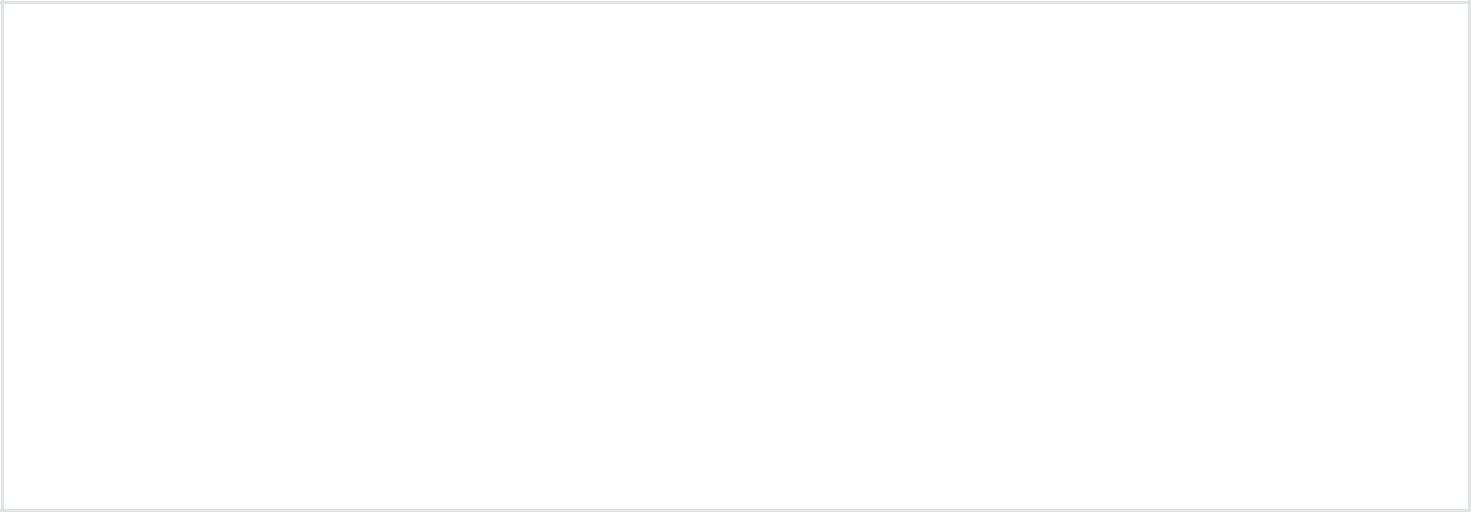
files

Define uma lista de arquivos que devem exis r na pasta do so ware para que possa ser executado.

paths

Define uma lista de pastas que devem exis r na pasta do so ware 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\_pan lt

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

controle\_pan lt

Objeto json que contém as configurações do pan- lt.

comunicacao\_serial

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

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 so ware.

video\_camera

Define um objeto de configuração da classe videoCamera.

port\_com

Define a porta de comunicação com o arduino.

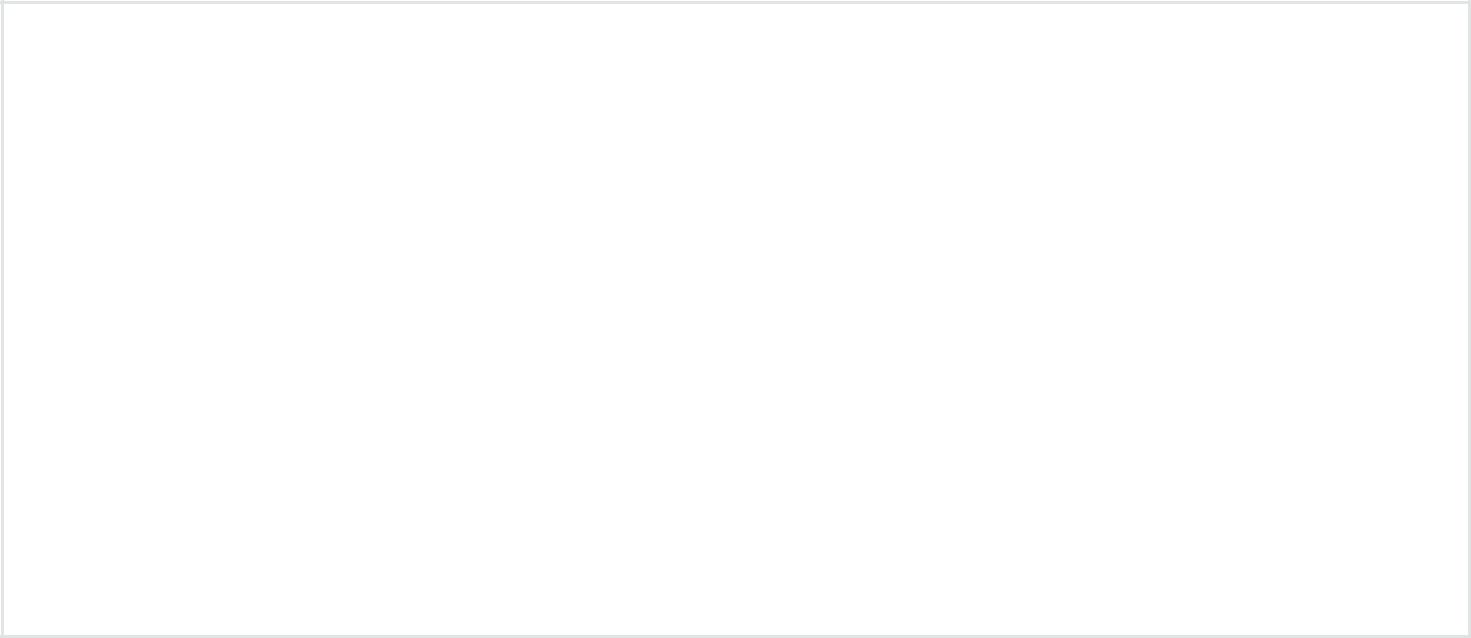
baudrate

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[¶](#page6)

Inicia todos os serviços da Estação LEONA.



main.camera(arg) [[código fonte]](file:///C:/Users/LEONA/Documents/Projetos Python/Projeto Leona/LEONA/LEONA v3 Tornado/LEONA TORNADO v3.0/documentacao/_modules/main.html" \l "camera)

Recebe comando para iniciar servidores

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

main.main() [[código fonte]](file:///C:/Users/LEONA/Documents/Projetos Python/Projeto Leona/LEONA/LEONA v3 Tornado/LEONA TORNADO v3.0/documentacao/_modules/main.html" \l "main)

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



main.verificar\_arquivos(files) [[código fonte]](file:///C:/Users/LEONA/Documents/Projetos Python/Projeto Leona/LEONA/LEONA v3 Tornado/LEONA TORNADO v3.0/documentacao/_modules/main.html" \l "verificar_arquivos)

Verifica se todos os arquivos necessários para execução do so ware existem

Parâmetros: files (json.array) – Lista json contendo os arquivos que deverão exis r na pasta do so ware



main.verificar\_dependencias(dependencies) [[código fonte]](file:///C:/Users/LEONA/Documents/Projetos Python/Projeto Leona/LEONA/LEONA v3 Tornado/LEONA TORNADO v3.0/documentacao/_modules/main.html" \l "verificar_dependencias)

Verifica se todos as dependências necessárias para execução do so ware existem

Parâmetros: dependencies (json.array) – Lista json contendo as dependências que deverão exis r para execução do so ware



main.verificar\_pastas(paths) [[código fonte]](file:///C:/Users/LEONA/Documents/Projetos Python/Projeto Leona/LEONA/LEONA v3 Tornado/LEONA TORNADO v3.0/documentacao/_modules/main.html" \l "verificar_pastas)

Verifica se todos as pastas necessárias para execução do so ware existem

Parâmetros: paths (json.array) – Lista json contendo as pastas que deverão exis r na pasta do so ware

servidor\_tornado\_camera\_1[¶](#page7)

Classe responsável pela transmissão das imagens.



|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| class servidor\_tornado\_camera\_1.ApiHandler(applica on, request, \*\*kwargs) | | | | | | | | [[código fonte]](file:///C:/Users/LEONA/Documents/Projetos Python/Projeto Leona/LEONA/LEONA v3 Tornado/LEONA TORNADO v3.0/documentacao/_modules/servidor_tornado_camera_1.html" \l "ApiHandler) |
| Base: | |  | | | |  |  |  |
| tornado.web.RequestHandler | | | |  |
|  | get(\*args) | | | [[código fonte]](file:///C:/Users/LEONA/Documents/Projetos Python/Projeto Leona/LEONA/LEONA v3 Tornado/LEONA TORNADO v3.0/documentacao/_modules/servidor_tornado_camera_1.html" \l "ApiHandler.get) |  |  |  |  |
|  |  |  |  |  |
|  | post() | | [[código fonte]](file:///C:/Users/LEONA/Documents/Projetos Python/Projeto Leona/LEONA/LEONA v3 Tornado/LEONA TORNADO v3.0/documentacao/_modules/servidor_tornado_camera_1.html" \l "ApiHandler.post) | |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  | | | | | | |  |
|  | | | | | | | |  |
| class servidor\_tornado\_camera\_1.SocketHandler(applica on, request, \*\*kwargs) | | | | | | | | [[código fonte]](file:///C:/Users/LEONA/Documents/Projetos Python/Projeto Leona/LEONA/LEONA v3 Tornado/LEONA TORNADO v3.0/documentacao/_modules/servidor_tornado_camera_1.html" \l "SocketHandler) |
| Base: | |  | | | | |  |  |
| tornado.websocket.WebSocketHandler | | | | |  |
|  | check\_origin(origin) | | | | [[código fonte]](file:///C:/Users/LEONA/Documents/Projetos Python/Projeto Leona/LEONA/LEONA v3 Tornado/LEONA TORNADO v3.0/documentacao/_modules/servidor_tornado_camera_1.html" \l "SocketHandler.check_origin) | | |  |
|  |  |
|  | on\_close() | | | [[código fonte]](file:///C:/Users/LEONA/Documents/Projetos Python/Projeto Leona/LEONA/LEONA v3 Tornado/LEONA TORNADO v3.0/documentacao/_modules/servidor_tornado_camera_1.html" \l "SocketHandler.on_close) | | | |  |
|  |  |
|  |  |
|  | open() | | [[código fonte]](file:///C:/Users/LEONA/Documents/Projetos Python/Projeto Leona/LEONA/LEONA v3 Tornado/LEONA TORNADO v3.0/documentacao/_modules/servidor_tornado_camera_1.html" \l "SocketHandler.open) | |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |

Inicia o envio de dados

render() [[código fonte]](file:///C:/Users/LEONA/Documents/Projetos Python/Projeto Leona/LEONA/LEONA v3 Tornado/LEONA TORNADO v3.0/documentacao/_modules/servidor_tornado_camera_1.html" \l "SocketHandler.render)

Prepara imagens para transmissão



writer= None



servidor\_tornado\_camera\_1.config

* {'is\_pan lt': '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 (se ngs.Se ngs):

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



servidor\_tornado\_camera\_1.queue= <mul processing.queues.Queue object>

Variável queue: inicia a pilha para armazenar imagens. Paramêtro: maxsize(330)–> Controla o buﬀer máximo de leitura da câmers, 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[¶](#page8)



|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| class servidor\_tornado\_camera\_2.ApiHandler(applica on, request, \*\*kwargs) | | | | | | | | [[código fonte]](file:///C:/Users/LEONA/Documents/Projetos Python/Projeto Leona/LEONA/LEONA v3 Tornado/LEONA TORNADO v3.0/documentacao/_modules/servidor_tornado_camera_2.html" \l "ApiHandler) |
| Base: | |  | | | |  |  |  |
| tornado.web.RequestHandler | | | |  |
|  | get(\*args) | | | [[código fonte]](file:///C:/Users/LEONA/Documents/Projetos Python/Projeto Leona/LEONA/LEONA v3 Tornado/LEONA TORNADO v3.0/documentacao/_modules/servidor_tornado_camera_2.html" \l "ApiHandler.get) |  |  |  |  |
|  |  |  |  |  |
|  | post() | | [[código fonte]](file:///C:/Users/LEONA/Documents/Projetos Python/Projeto Leona/LEONA/LEONA v3 Tornado/LEONA TORNADO v3.0/documentacao/_modules/servidor_tornado_camera_2.html" \l "ApiHandler.post) | |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  | | | | | | |  |
|  | | | | | | | |  |
| class servidor\_tornado\_camera\_2.SocketHandler(applica on, request, \*\*kwargs) | | | | | | | | [[código fonte]](file:///C:/Users/LEONA/Documents/Projetos Python/Projeto Leona/LEONA/LEONA v3 Tornado/LEONA TORNADO v3.0/documentacao/_modules/servidor_tornado_camera_2.html" \l "SocketHandler) |
| Base: | |  | | | | |  |  |
| tornado.websocket.WebSocketHandler | | | | |  |
|  | check\_origin(origin) | | | | [[código fonte]](file:///C:/Users/LEONA/Documents/Projetos Python/Projeto Leona/LEONA/LEONA v3 Tornado/LEONA TORNADO v3.0/documentacao/_modules/servidor_tornado_camera_2.html" \l "SocketHandler.check_origin) | | |  |
|  |  |
|  | on\_close() | | | [[código fonte]](file:///C:/Users/LEONA/Documents/Projetos Python/Projeto Leona/LEONA/LEONA v3 Tornado/LEONA TORNADO v3.0/documentacao/_modules/servidor_tornado_camera_2.html" \l "SocketHandler.on_close) | | | |  |
|  |  |
|  |  |
|  | open() | | [[código fonte]](file:///C:/Users/LEONA/Documents/Projetos Python/Projeto Leona/LEONA/LEONA v3 Tornado/LEONA TORNADO v3.0/documentacao/_modules/servidor_tornado_camera_2.html" \l "SocketHandler.open) | |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |

Inicia o envio de dados

render() [[código fonte]](file:///C:/Users/LEONA/Documents/Projetos Python/Projeto Leona/LEONA/LEONA v3 Tornado/LEONA TORNADO v3.0/documentacao/_modules/servidor_tornado_camera_2.html" \l "SocketHandler.render)

Prepara imagens para transmissão



writer= None



servidor\_tornado\_camera\_2.config

* {'is\_pan lt': '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 (se ngs.Se ngs):

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



servidor\_tornado\_camera\_2.queue= <mul processing.queues.Queue object>

Variável queue: inicia a pilha para armazenar imagens. Paramêtro: maxsize(330)–> Controla o buﬀer máximo de leitura da câmers, 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[¶](#page9)



class video\_camera.VideoCamera(path=0, queue=None) [[código fonte]](file:///C:/Users/LEONA/Documents/Projetos Python/Projeto Leona/LEONA/LEONA v3 Tornado/LEONA TORNADO v3.0/documentacao/_modules/video_camera.html" \l "VideoCamera)

Base: threading.Thread



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

Inicia a câmera

Parâmetros:



path (inteiro) – Indica qual placa de captura deve ser u liza. Número iden do

ao indicado na placa conectada a placa mãe da Estação Leona.

queue (mul processing.Queue) – array reponsável por guardar os frames

para que outro processo possa ler

out= None

Variável out: (cv2.VideoWriter) Instância de VideoWriter



queue= None

Variável queue: (mul processing.Queue) Instância de Queue

recorded= None

Variável recorded:

(boolean) Controla inicialização e finalização da gravação



run() [[código fonte]](file:///C:/Users/LEONA/Documents/Projetos Python/Projeto Leona/LEONA/LEONA v3 Tornado/LEONA TORNADO v3.0/documentacao/_modules/video_camera.html" \l "VideoCamera.run)

Métodod executado pela Thread



start\_record() [[código fonte]](file:///C:/Users/LEONA/Documents/Projetos Python/Projeto Leona/LEONA/LEONA v3 Tornado/LEONA TORNADO v3.0/documentacao/_modules/video_camera.html" \l "VideoCamera.start_record)

Inicia a gravação em disco das imagens capturadas pela câmera



stop() [[código fonte]](file:///C:/Users/LEONA/Documents/Projetos Python/Projeto Leona/LEONA/LEONA v3 Tornado/LEONA TORNADO v3.0/documentacao/_modules/video_camera.html" \l "VideoCamera.stop)

Finaliza a leitura da camera



stop\_record() [[código fonte]](file:///C:/Users/LEONA/Documents/Projetos Python/Projeto Leona/LEONA/LEONA v3 Tornado/LEONA TORNADO v3.0/documentacao/_modules/video_camera.html" \l "VideoCamera.stop_record)

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[¶](#page10)

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 mul processing.Queue

Debug

Osciloscopio

Permite o monitoramento do loop com osciloscopio 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 comunicacaoserial

Arquivo txt

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



class video\_camera\_teste.VideoCamera(path=0, queue=None, com=None, arquivo=None)

[[código fonte]](file:///C:/Users/LEONA/Documents/Projetos Python/Projeto Leona/LEONA/LEONA v3 Tornado/LEONA TORNADO v3.0/documentacao/_modules/video_camera_teste.html" \l "VideoCamera)

Base: threading.Thread



Inicia a câmera

Parâmetros:



path (String) – recebe caminho de leitura da câmera

queue (mul processing.Queue) – array reponsável por guardar os frames

para que outro processo possa ler

com (comunicacaoserial.ComunicacaoSerial) – recebe a com reponsável 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: (mul processing.Queue) Instância de Queue



recorded= None

Variável recorded:

(boolean) Controla inicialização e finalização da gravação



run() [[código fonte]](file:///C:/Users/LEONA/Documents/Projetos Python/Projeto Leona/LEONA/LEONA v3 Tornado/LEONA TORNADO v3.0/documentacao/_modules/video_camera_teste.html" \l "VideoCamera.run)

Métodod executado pela Thread



start\_record() [[código fonte]](file:///C:/Users/LEONA/Documents/Projetos Python/Projeto Leona/LEONA/LEONA v3 Tornado/LEONA TORNADO v3.0/documentacao/_modules/video_camera_teste.html" \l "VideoCamera.start_record)

Inicia a gravação em disco das imagens capturadas pela câmera



stop() [[código fonte]](file:///C:/Users/LEONA/Documents/Projetos Python/Projeto Leona/LEONA/LEONA v3 Tornado/LEONA TORNADO v3.0/documentacao/_modules/video_camera_teste.html" \l "VideoCamera.stop)

Finaliza a leitura da camera



stop\_record() [[código fonte]](file:///C:/Users/LEONA/Documents/Projetos Python/Projeto Leona/LEONA/LEONA v3 Tornado/LEONA TORNADO v3.0/documentacao/_modules/video_camera_teste.html" \l "VideoCamera.stop_record)

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[¶](#page12)

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



class comunicacao\_serial.ComunicacaoSerial(port='COM3', baudrate=9600) [[código fonte]](file:///C:/Users/LEONA/Documents/Projetos Python/Projeto Leona/LEONA/LEONA v3 Tornado/LEONA TORNADO v3.0/documentacao/_modules/comunicacao_serial.html" \l "ComunicacaoSerial)

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 baudarate:

Recebe a taxa de aquisição de dados feita pelo Arduino



enviar(arg) [[código fonte]](file:///C:/Users/LEONA/Documents/Projetos Python/Projeto Leona/LEONA/LEONA v3 Tornado/LEONA TORNADO v3.0/documentacao/_modules/comunicacao_serial.html" \l "ComunicacaoSerial.enviar)

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

Parâmetros: arg (inteiro) – Dado para teste da taxa de leitura das cameras. Teste feito

com Osiloscópio 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\_pan lt[¶](#page13)

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



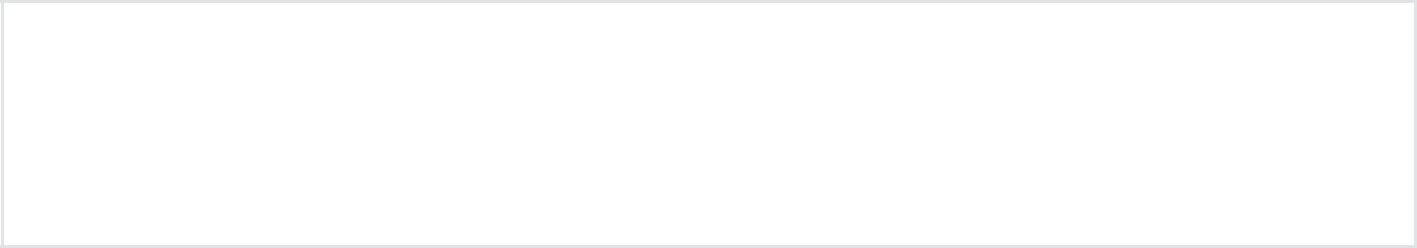
class controle\_pantilt.ControlePantilt(con=None) [[código fonte]](file:///C:/Users/LEONA/Documents/Projetos Python/Projeto Leona/LEONA/LEONA v3 Tornado/LEONA TORNADO v3.0/documentacao/_modules/controle_pantilt.html" \l "ControlePantilt) Base: object



Iniciar a posição do Pan-Tilt

Parâmetros: con (object) – Instância de comunicao 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

!111O\* - Ligar camera (um,um,um,zero)

!111F\* - Desligar camera



Valores máximos e mínimos:

Azimute: 0º -> 350º Elevação: -35º -> +35º



azGraus= None

Variável azGraus:

(inteiro) Armazena a ul ma posicao do pan- lt em azimute



azimute(graus) [[código fonte]](file:///C:/Users/LEONA/Documents/Projetos Python/Projeto Leona/LEONA/LEONA v3 Tornado/LEONA TORNADO v3.0/documentacao/_modules/controle_pantilt.html" \l "ControlePantilt.azimute)

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 azimute



con= None

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



converterelevacao(graus) [[código fonte]](file:///C:/Users/LEONA/Documents/Projetos Python/Projeto Leona/LEONA/LEONA v3 Tornado/LEONA TORNADO v3.0/documentacao/_modules/controle_pantilt.html" \l "ControlePantilt.converterelevacao)

Converte o valor recebido para o valor esperado pelo controlador de Pan-Tilt, onde -35º == 0, 0º == 35, 35º == 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]](file:///C:/Users/LEONA/Documents/Projetos Python/Projeto Leona/LEONA/LEONA v3 Tornado/LEONA TORNADO v3.0/documentacao/_modules/controle_pantilt.html" \l "ControlePantilt.descer)

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]](file:///C:/Users/LEONA/Documents/Projetos Python/Projeto Leona/LEONA/LEONA v3 Tornado/LEONA TORNADO v3.0/documentacao/_modules/controle_pantilt.html" \l "ControlePantilt.desligarcamera)

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



direita(graus) [[código fonte]](file:///C:/Users/LEONA/Documents/Projetos Python/Projeto Leona/LEONA/LEONA v3 Tornado/LEONA TORNADO v3.0/documentacao/_modules/controle_pantilt.html" \l "ControlePantilt.direita)

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 ul ma posicao do pan- lt em elevacao



elevacao(graus) [[código fonte]](file:///C:/Users/LEONA/Documents/Projetos Python/Projeto Leona/LEONA/LEONA v3 Tornado/LEONA TORNADO v3.0/documentacao/_modules/controle_pantilt.html" \l "ControlePantilt.elevacao)

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]](file:///C:/Users/LEONA/Documents/Projetos Python/Projeto Leona/LEONA/LEONA v3 Tornado/LEONA TORNADO v3.0/documentacao/_modules/controle_pantilt.html" \l "ControlePantilt.esquerda)

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]](file:///C:/Users/LEONA/Documents/Projetos Python/Projeto Leona/LEONA/LEONA v3 Tornado/LEONA TORNADO v3.0/documentacao/_modules/controle_pantilt.html" \l "ControlePantilt.formatargraus)

Verifica valor informado e formata no padrão de bits

Parâmetros:



graus (inteiro) – valor de azimute ou elevação selecionado pelo observador da Rede Leona

ligarcamera()

[[código fonte]](file:///C:/Users/LEONA/Documents/Projetos Python/Projeto Leona/LEONA/LEONA v3 Tornado/LEONA TORNADO v3.0/documentacao/_modules/controle_pantilt.html" \l "ControlePantilt.ligarcamera)

Liga as câmeras:



resetarazimute()

[[código fonte]](file:///C:/Users/LEONA/Documents/Projetos Python/Projeto Leona/LEONA/LEONA v3 Tornado/LEONA TORNADO v3.0/documentacao/_modules/controle_pantilt.html" \l "ControlePantilt.resetarazimute)

Envia para a controladora !350L, assim resetando azimute em 0º



resetarelevacao() [[código fonte]](file:///C:/Users/LEONA/Documents/Projetos Python/Projeto Leona/LEONA/LEONA v3 Tornado/LEONA TORNADO v3.0/documentacao/_modules/controle_pantilt.html" \l "ControlePantilt.resetarelevacao)

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



subir(graus) [[código fonte]](file:///C:/Users/LEONA/Documents/Projetos Python/Projeto Leona/LEONA/LEONA v3 Tornado/LEONA TORNADO v3.0/documentacao/_modules/controle_pantilt.html" \l "ControlePantilt.subir)

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

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

Gerar automa camente a documentação de sofware[¶](#page15)

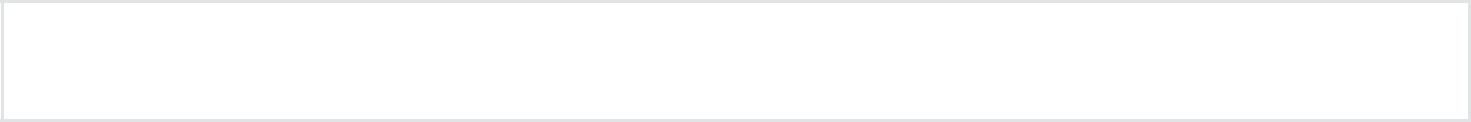
A documentação da Estação LEONA v3.0 foi gerada automa camente com o uso de biblioteca Sphinx, para u lizar o mesmo método com esta ou outras versões do so ware em python basta seguir os passos abaixo.

Requisitos[¶](#page15)

Para u lizar a biblioteca Sphinx é necessário que a versão 3.6.1 ou superior do python esteja [instalada. Para instalar você pode checar a seção So ware - Detalhamento das etapas para](file:///C:/Users/LEONA/Documents/Projetos Python/Projeto Leona/LEONA/LEONA v3 Tornado/LEONA TORNADO v3.0/documentacao/Software.html" \l "detalhamento-das-etapas-para-configuracao-do-software-da-estacao-leona.html) [configuração do So ware da Estação LEONA](file:///C:/Users/LEONA/Documents/Projetos Python/Projeto Leona/LEONA/LEONA v3 Tornado/LEONA TORNADO v3.0/documentacao/Software.html" \l "detalhamento-das-etapas-para-configuracao-do-software-da-estacao-leona.html)

Preparando o esquema básico de documentação[¶](#page15)

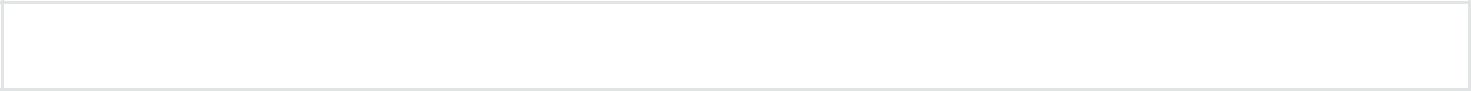
Instalar:



pip install sphinx

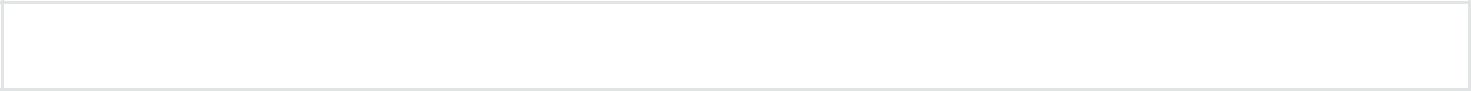
pip install sphinx\_rtd\_theme

Na pasta dos scripts execute o comando:



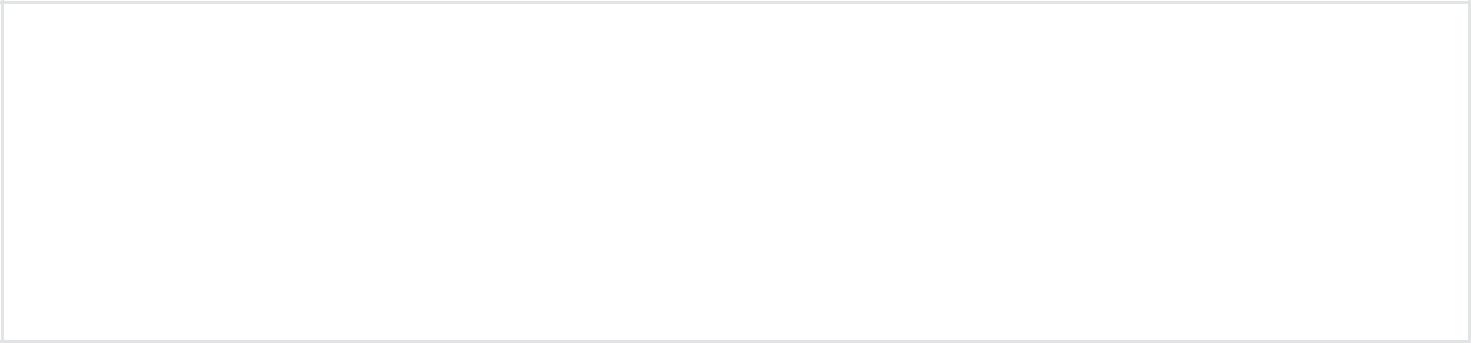
md documentacao scripts rst

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

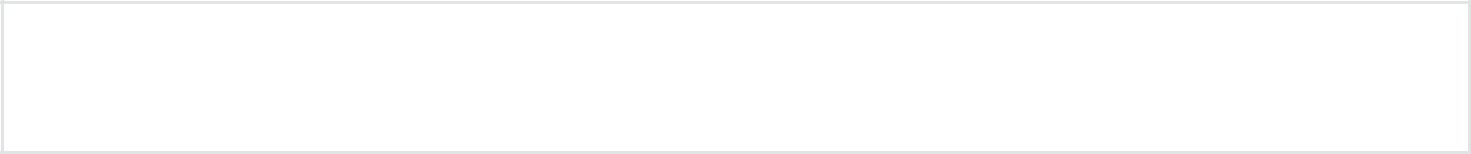
viewcode[y]: y

Create Makefile[y]: y

Create Windows command file[y]: y

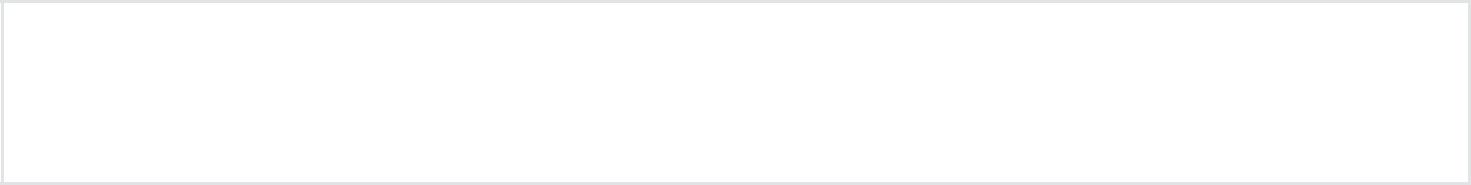
No arquivo conf.py modifique as linhas:

Subs tua as linhas:



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

Pelas linhas:



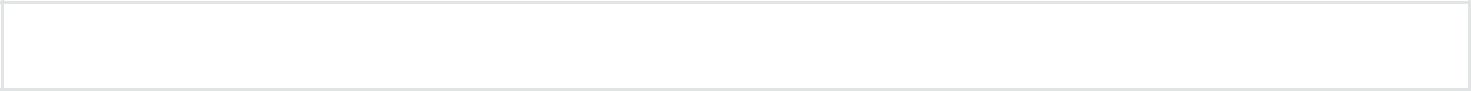
import os

import sys

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

import sphinx\_rtd\_theme

Para garan r que a a def \_\_init\_\_ será automa camente documentada adicione a linha:

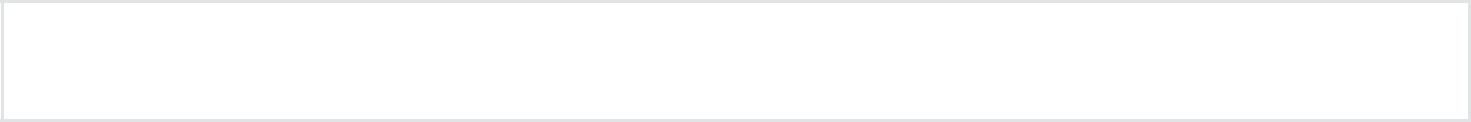


autoclass\_content = 'both'

Na seção # – General configura on —

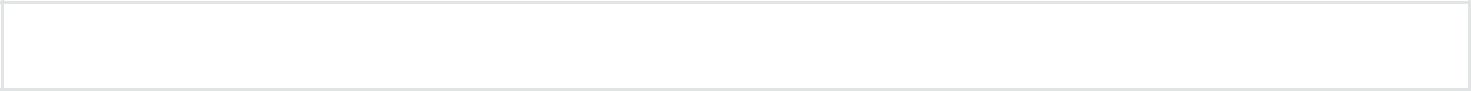


Na linha:



* The master toctree document master\_doc = 'index'

Altere ‘index’ para:

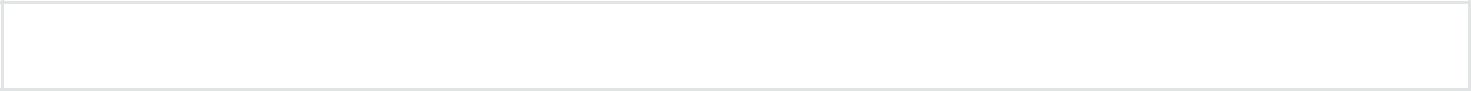


master\_doc = 'RedeLeonaDoc'

Na linha # – Op ons for HTML output —–#

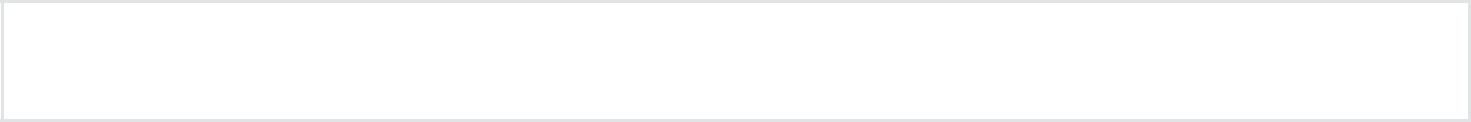


Comente:



html\_theme = 'alabaster'

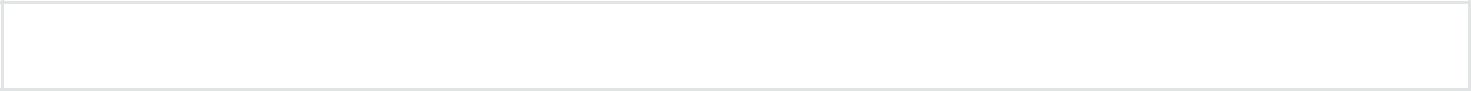
E insira:



html\_theme = "sphinx\_rtd\_theme"

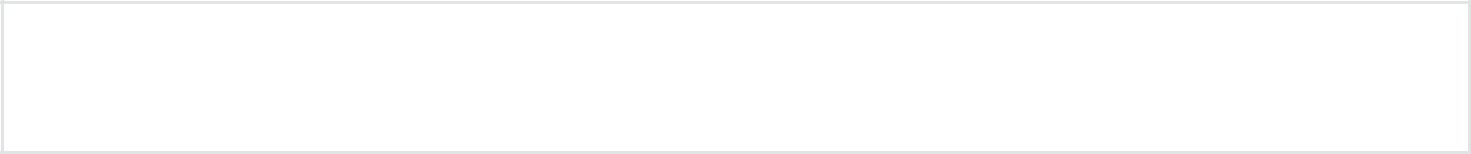
html\_theme\_path = [sphinx\_rtd\_theme.get\_html\_theme\_path()]

Execute o comando:



make html

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



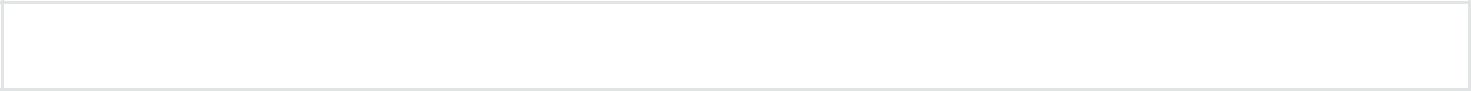
sphinx-apidoc -o ./rst ./scripts

cd rst

md \_static

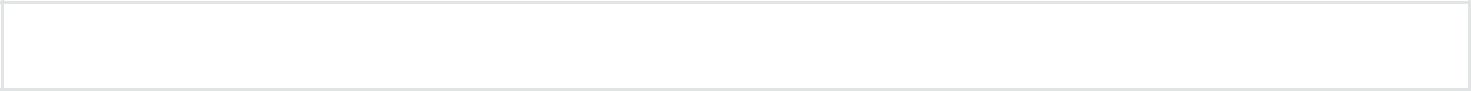
Recorte o script “conf.py” e “index.rst” para a pasta criada “rst”.

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[¶](#page18)

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 se ngs.json não é automa camente documentado, qualquer alteração

deve ser feita diretamente no arquivo ‘se ngs.rst’, seção: ‘Arquivo de configuração se ngs.json’.