# Drone Autônomo

**Missões autônomas e manutenção de posição**

*Rafael Farias Meneses*

*None*

# Tabela de conteúdos

# 1. Home

## 1.1 Apresentação

Esta é a documentação do projeto **Desenvolvimento de veículo autônomo não tripulado para realização de missões de monitorização de parâmetros topográficos e de qualidade da água de rios e lagos**, uma colaboração entre a **Associação de Pesquisa Desenvolvimento e Tecnologia Hefestus** e a **Universidade Tecnológica Federal do Paraná - UTFPR** *campus* Toledo.

Esta documentação gera automaticamente um arquivo em PDF de seu conteúdo, este pode ser baixado aqui.

> Gostou dessa documentação? visite o repositório ZRafaF/ReadTheDocksBase para mais informações 😄.

Utilize as ***Tabs*** a cima para navegar pelo site.

## 1.2 Setup

Este projeto assume que você possui a versão mais recente de Python3, de **PIP** e **GIT**, caso precise instalar por favor visite https://www.python.org/downloads/.

Este projeto foi testado e desenvolvido com a versão `Python 3.10.x`

### 1.2.1 Clonando o repositório

**Bash**

```
git clone https://github.com/ZRafaF/OpencvPosHold

cd OpencvPosHold
```

### 1.2.2 Criando ambiente virtual

Esse passo não é obrigatório, mas sim **recomendado**

**Bash**

```
python3 -m pip install --user virtualenv

python -m venv venv
```

Com isso um ambiente virtual chamado `venv` será criado no diretório do projeto.

Para ativar:

**Ativação no Windows**

```
venv/Scripts/activate
```

ou

**Ativação no Linux**

```
source venv/bin/activate
```

### 1.2.3 Instalando dependências

Primeiro será necessário instalar o OpenCV, neste caso temos 2 opções de instalação, utilizar um **gerenciador de pacotes** ou compilar da fonte.

Em sistemas embarcados, como a Raspberry PI, é recomendado que este seja **compilado da fonte**

> **Warning**
>
> Devido a evolução do projeto com o tempo a versão do OpenCV que está sendo utilizada é a `opencv_contrib`, entretanto tecnicamente não estamos utilizando nenhum pacote da versão contrib. Logo TALVEZ seja possível utilizar apenas a versão padrão do OpenCV.

## OpenCV

A versão que estaremos instalando é a `opencv_contrib`, para informações sobre a instalação da versão padrão por favor verificar a documentação.

> **Utilizando um gerenciador de pacotes**
>
> ```
> pip install opencv-contrib-python
> ```
>
> Caso deseje a opção *headless* pode usar
>
> ```
> pip install opencv-contrib-python-headless
> ```

ou

> **Compilando da fonte**
>
> 1. Seguir o tutorial de instalação (cerca de 8 horas na RaspberryPi 3 ) aqui
> 2. Linkar o modulo aqui, vá na categoria *Sym-link your OpenCV 4 on the Raspberry Pi*
>
> Por fim será necessário linkar-lo ao projeto:
>
> **Bash**
>
> ```
> cd /usr/local/lib/python3.9/site-packages/cv2/python-3.9
>
> sudo mv cv2.cpython-39-arm-linux-gnueabihf.so cv2.so
>
> cd <OpencvPosHold>
>
> ln -s /usr/local/lib/python3.9/site-packages/cv2/python-3.9/
>
> cv2.so cv2.so
> ```

## Outros

**Bash**

```
pip install numpy

sudo pip install pupil-apriltags

sudo pip install dronekit

pip install pymavlink

pip install argparse

pip install imutils

pip install picamera
```

# 2. Utilização

## 2.1 Utilização

Aqui serão apresentados os métodos de utilização deste software

## 2.2 Calibração da câmera

Para a aquisição de leituras e estimação de distâncias é imprescindível uma boa calibração da câmera.

A camera utilizada neste projeto foi a Raspberry Pi Camera Module 2.

Este projeto possui scripts de Python para calibrar a câmera. O processo de calibração segue os seguinte passos:

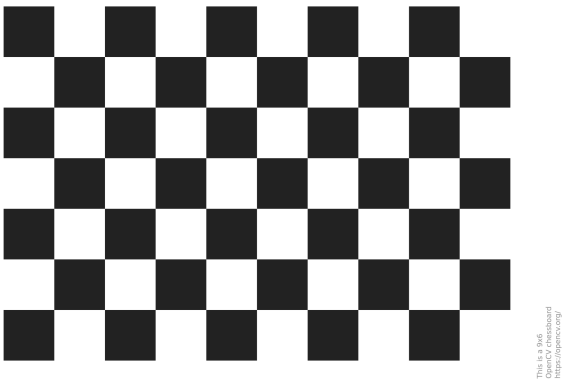
1. `CaptureImg.py` : Este scripts irá tirar fotos e armazenar as mesmas.
2. `aprilCalibrateCam.py` : Este script realizará os cálculos e irá gerar uma **matriz de distorção**.

> Esses scripts foram derivados do repositório Basic-Augmented-reality-course-opencv, por favor o visite e deem as atribuições necessárias.

### 2.2.1 Capturando imagens

Siga os passos a seguir para capturar as imagens:

1. Imprima o chessboard pattern.



2. Deixe **plano** fixando-o à alguma superfície.

   Por exemplo um pedaço de madeira, isopor ou qualquer outro material plano.

3. Coloque as **dimensões** no script `CaptureImg.py`

**Python**

```
# Dimensão do tabuleiro
CHESS_BOARD_DIM = (9, 6)
```

Note que o numero começa em 0, ou seja, se o *pattern* possui 10 x 7, você colocará `CHESS_BOARD_DIM = (9, 6)` .

4. Inicie o script e pressione `s` para salvar uma foto, ou `q` para finalizar o programa.

- É importante que sejam tiradas fotos com o *pattern* em posições e inclinações diferentes, principalmente nas **bordas** do campo de visão da câmera;
- Para um bom resultado recomendo tirar pelo menos **30 fotos diferentes**.

Ao finalizar o programa você terá um diretório chamado `/images/` na *root* do projeto.

---

## 2.2.2 Gerando matriz

Neste passo será gerado a matriz de calibração ela irá gerar uma matriz que você deverá **copiar** e coloca-lo no script principal.

Siga as instruções a seguir para realizar a calibração com sucesso.

# Setup

- Abra o arquivo `aprilCalibrateCam.py` em seu editor de preferencia.
- O primeiro passo é colocar os parâmetros do *chessboard pattern*. Altere os seguintes parâmetros:

**Python**

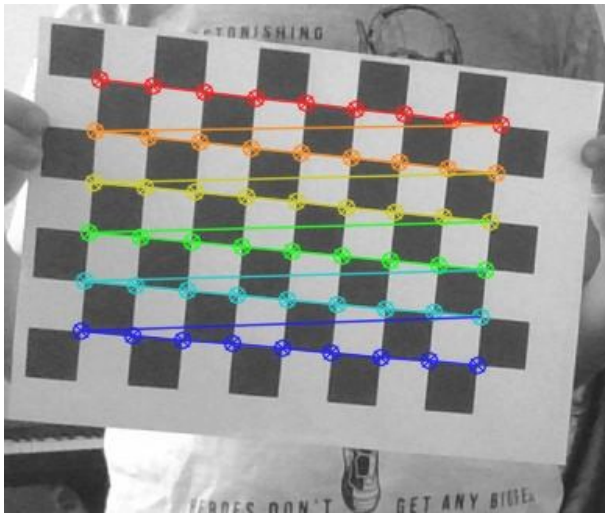
```
CHESS_BOARD_DIM = (9, 6)
SQUARE_SIZE = 26  # - mm
```

- `CHESS_BOARD_DIM` deverá conter os mesmos valores dos colocados no item anterior.
- `SQUARE_SIZE` deverá conter o tamanho em `mm` dos quadrados do seu *pattern*.

# Apuração

- Após o setup das constantes o próximo passo será **apurar** as imagens. Ao iniciar o arquivo serão apresentadas as imagens que foram capturadas com as linhas de calibração.

Exemplo:



O exemplo acima é uma base de como as suas imagens devem estar, sem linhas cruzadas e com todos os cantos identificados.

- O script irá iterar sobre todas as imagens no diretório `/images/`
- Você devera pressionar `s` para **aceitar** a imagem, `q` para **encerrar** ou qualquer outra letra para **rejeitar**.

Ao fim será impresso na tela uma matriz como a seguinte:

```
cam_params = (643.2857240774916, 644.3167085172134, 336.7788196007684, 231.26676197077208)
```

Você deverá copiar essa linha para adiciona-la ao script principal posteriormente.

> **Warning**
>
> O plano inicial era gerar um arquivo `.npz` e guardar a matriz lá, entretanto essa *feature* não foi implementada ainda.

## 2.2.3 Aplicando calibração

Para aplicar a calibração siga os passo a seguir:

1. Abra o script principal `2aprilTagDetect.py`
2. Altere a variável `cam_params` com seus valores próprios

**Python**

```
# Parametros gerados do script aprilCalibrateCam.py
cam_params = (
    630.8669379442165,
    630.3123204518172,
    335.75042566981904,
    227.83332282734318,
)
```

## 2.3 Missão

Aqui será apresentado como utilizar o software em uma missão real

## 2.4 Simulação

Aqui será apresentado como executar simulações de missões.

# 3. Hardware

## 3.1 Hardware

Esta documentação fara um *overview* do *hardware* utilizado, também dará instruções e informações para auxiliar na reprodução dos resultados.

## 3.2 Controladora de voo

### 3.2.1 Mini Pix

A controladora escolhida foi a Mini Pix V1.0. Este modelo é baseado na arquitetura da PixHawk.

> **Warning**
>
> Cuidado na escolha da controladora, este modelo foi escolhido após estudo profundo do firmware que seria usado o ArduPilot. E a sua documentação conta com um aviso sobre esse modelo de controladora, falando basicamente:
>
> Algumas versões dessa controladora não são **recomendadas** devido a uma possivel incompatibilidade.
>
> “V1.0” e “V1.2” provavelmente funcionam, “V1.0 II” e “V1.1” definitivamente não funcionam.
>
> Essa documentação pode ser encontrada aqui.

### 3.2.2 ArduPilot

O firmware ArduPilot foi escolhido devido a sua flexibilidade e facilidade de integração com os outros componentes requeridos pelo projeto.

## 3.3 Frame

Durante todo o desenvolvimento sempre foram utilizados 2 tipos de frames.

### 3.3.1 **Frame de validação**

- Utilizado para testar tecnologias;
- Fácil acesso para facilitar a iteração e modificação de hardware;
- Foi utilizado o frame **s500** da dji. Uma plataforma um pouco defasada a este ponto, entretanto com bastante material disponível.

### 3.3.2 **Frame final**

Ao decorrer do projeto o **frame aprova d'água** foi alterado por diversas vezes. Os notáveis serão apresentados a seguir.

## QuadH2o

O projeto teve como seu frame inicial o QuadH2o, entretanto devido a sua descontinuação se tornou inviável para o projeto.

## Custom

Esse foi o **frame final**, o desenho foi feito utilizando o Fusion 360 e seu `.step` está disponível em EM BREVE.

Seu corpo principal foi impresso em uma impressora 3d e canibalizou os braços e trem de pouso da s500.

## 3.3.3 Desenvolvimento

Este capitulo descreverá o processo de desenvolvimento e fabricação do frame customizado, para que terceiros sejam capazes de reproduzi-lo.

# 4. Software

## 4.1 software

Esta documentação fara um *overview* do *software* utilizado, também dará instruções e informações para auxiliar na reprodução dos resultados.

## 4.2 API

### 4.2.1 api

# 5. Sobre

## 5.1 Sobre

Esta página contem informações adicionais e as licenças de softwares de terceiros.