

PUMATRONIX

Classifier

Manual do Usuário

Biblioteca de Software para Reconhecimento Automático de Características Veiculares

Release: v1.5.0
Data: 27/10/2020

Sumário

- **Histórico de alterações**
- 1. Visão Geral
 - 1.1. Condições Gerais
 - 1.2. Licença de software
- 2. Introdução
 - 2.1. Casos de Uso
 - 2.2. Princípio de Funcionamento
 - 2.3. Limitações de Uso
 - 2.3.1. Uso em máquinas virtuais (VMs)
- 3. Guia de Uso
 - 3.1. Condições de Uso
 - 3.2. Instalação
 - 3.2.1. Pré-requisitos do sistema
 - 3.2.2. Descompactando o SDK
 - 3.2.3. Permissões do *hardkey*
 - 3.2.4. Variáveis de ambiente
 - 3.3. Estrutura do SDK
 - 3.3.1. Árvore de arquivos versão Linux
 - 3.3.2. Árvore de arquivos versão Windows
 - 3.4. Arquitetura de software
 - 3.5. Exemplo de uso da API
- 4. APIs de usuário
 - 4.1. API Classifier C/C++
 - 4.1.1. API Classifier C/C++
 - 4.2. API ptx_common C/C++
 - 4.2.1. **ptx_codes.h**
 - 4.2.2. **ptx_dict.h**
 - 4.2.3. **ptx_image.h**

Histórico de alterações

Data	Versão	Revisão
10/06/2019	1.0.0	<ul style="list-style-type: none">• Versão inicial
12/06/2019	1.1.0	<ul style="list-style-type: none">• Adição de wrapper Java• Adição de API para consultar informações da licença
01/07/2019	1.2.0	<ul style="list-style-type: none">• Melhoria da classificação de tipo de veículo para o cenário fechado• Adição de API para configurar a probabilidade mínima do classificador de tipo de veículo
26/07/2019	1.3.0	<ul style="list-style-type: none">• Melhoria da classificação de tipo de veículo para o cenário fechado• Redução de tempo de processamento para o cenário fechado• Correção da probabilidade de saída
10/12/2019	1.4.0	<ul style="list-style-type: none">• Melhoria da classificação de tipo de veículo para o cenário panorâmico• Portabilidade para plataformas Windows 32 e 64 bits
27/10/2020	1.5.0	<ul style="list-style-type: none">• Adição do cenário panorâmico noturno (uso específico em imagens IR com baixa luminosidade)• Melhoria no índice de detecção do cenário panorâmico• Melhoria no índice de rejeição de falsos positivos

1. Visão Geral

1.1. Condições Gerais

Os dados e as informações contidas neste documento não podem ser alterados sem a permissão expressa por escrito da Pumatronix Equipamentos Eletrônicos. Nenhuma parte deste documento pode ser reproduzida ou transmitida para qualquer finalidade, seja por meio eletrônico ou físico.

Copyright © Pumatronix Equipamentos Eletrônicos. Todos os direitos reservados.

1.2. Licença de software

O software e a documentação em anexo estão protegidos por direitos autorais. Ao instalar o software, você concorda com as condições do contrato de licença.

2. Introdução

Classifier é uma biblioteca de software desenvolvida pela **Pumatronix Equipamentos Eletrônicos** especializada no reconhecimento de características de veículos a partir de imagens. Atualmente ela pode retornar o tipo do veículo dentre algumas categorias pré-definidas, sua localização na imagem e a confiabilidade da classificação.

Os tipos de veículos que a biblioteca atualmente pode distinguir são os seguintes:

- Carro
- Moto
- Caminhão
- Ônibus

O **Classifier** também pode detectar que não há nenhum veículo na imagem.

Após processar a imagem fornecida na entrada, o **Classifier** retorna as seguintes informações para cada veículo detectado:

- Tipo do veículo
- Coordenadas do veículo na imagem, na forma de um *bounding box* (retângulo)
- Confiabilidade da classificação de tipo veicular, na forma de uma probabilidade entre 0 e 1

2.1. Casos de Uso

A biblioteca **Classifier** foi desenvolvida para ser utilizada nas mais diversas aplicações que necessitam de informações sobre os veículos. Embora existam limitações técnicas (ver [Limitações de Uso](#)), o **Classifier** pode ser usado com imagens panorâmicas ou fechadas. Para aumentar seu desempenho, deve ser definido através de API o tipo de cena que será utilizado em seu funcionamento.

Alguns casos de uso mais comuns para a biblioteca **Classifier**, sem entretanto ficar restrita a eles, são:

- Classificação dos tipos de veículos em imagens
- Localização de veículos em imagens
- Detecção de imagens que contém veículos
- Validação do reconhecimento de placas veiculares através da classificação do tipo de veículo
- Levantamento de estatísticas de tipos de veículos através de imagens



Exemplos de aplicação do Classifier

Cima: imagens a serem processadas pelo Classifier

Baixo: imagens sobrepostas com informações obtidas pelo Classifier

Esquerda e centro: imagens do tipo fechada

Direita: imagem do tipo panorâmica

2.2. Princípio de Funcionamento

O **Classifier** opera detectando e classificando objetos em imagens. A partir desse processo é possível inferir as características de cada veículo na imagem separadamente.

2.3. Limitações de Uso

Para que o **Classifier** opere de forma satisfatória as condições a seguir devem ser atendidas.

2.3.1. Uso em máquinas virtuais (VMs)

Embora seja possível a utilização de nossas bibliotecas em máquinas virtuais, havendo inclusive relatos de sucesso por parte de alguns clientes, seu uso é desencorajado e não homologado. A **Pumatronix** não dá garantias de funcionamento e de suporte para o uso de seus produtos em máquinas virtuais.

3. Guia de Uso

Este capítulo traz as informações necessárias para integrar a biblioteca **Classifier** à uma aplicação.

3.1. Condições de Uso

O kit de desenvolvimento de software (SDK) do **Classifier** é distribuído através de um conjunto de bibliotecas de software (.so e .dll) com *interface de programação de aplicação* (API) em linguagem C. As arquiteturas de hardware compatíveis são x86, x86_64 e ARM A53 com sistema operacional Linux ou Windows. O suporte a outras linguagens de programação é possível através da escrita de *wrappers* ou *bindings* a partir da API C.

Para o correto funcionamento da biblioteca é necessário o uso do *hardkey* (chave de segurança) que a acompanha. O *hardkey* deverá estar conectado à uma porta USB do equipamento onde a aplicação será executada, caso contrário a aplicação não retornará as informações requisitadas. Existem duas versões de *hardkey*, uma de demonstração e outra para uso geral, sendo que a versão de demonstração tem data de validade. Quando a data de validade desta expira, a biblioteca automaticamente para de funcionar.

Para saber mais sobre o suporte em outras arquiteturas, como ARMv7, e outras linguagens como Java, Python ou C#, ou ainda solicitar a compra de uma licença entre em contato com a **Pumatronix** através do email contato@pumatronix.com.br.

3.2. Instalação

3.2.1. Pré-requisitos do sistema

3.2.1.1. Software

- Sistema operacional
 - Linux 32/64 bits com suporte a GLIBC 2.7 ou superior
 - Windows 32/64 bits versão 7 ou superior
- 7zip: <http://www.7-zip.org/>

3.2.1.2. Hardware

- Porta USB (utilizada pelo *hardkey*)
- CPU x86, x86_64 ou ARM A53
- 2GB ou mais de memória RAM

3.2.2. Descompactando o SDK

O SDK do **Classifier** é distribuído através de um arquivo compactado no formato 7zip. A descompactação deste arquivo requer uma senha, fornecida junto com o email contendo as instruções de download deste SDK. Caso você tenha problemas com a descompactação do SDK contate o suporte pelo email contato@pumatronix.com.br.

Para descompactar o SDK em ambiente Linux, utilize o seguinte comando a partir de um terminal (substitua os campos <Classifier_PC_LINUX_64_vX.Y.Z.7z> pela senha fornecida por email e pelo nome correto do pacote).

```
7z x -p<SENHA_FORNECIDA> <Classifier_PC_LINUX_64_vX.Y.Z.7z>
```

3.2.3. Permissões do *hardkey*

Esta configuração só é necessária para a versão Linux do SDK.

Para o correto funcionamento do *hardkey* USB no Linux, as permissões de acesso do **udev** devem ser alteradas. Adicione a seguinte linha:

```
ATTRS{idVendor}=="0403", ATTRS{idProduct}=="c580", MODE="0666"
```

ao final do arquivo correspondente a sua distribuição Linux:

```
Centos 5.2/5.4: /etc/udev/rules.d/50-udev.rules
Centos 6.0 em diante: /lib/udev/rules.d/50-udev-default.rules
Ubuntu 7.10: /etc/udev/rules.d/40-permissions.rules
Ubuntu 8.04/8.10: /etc/udev/rules.d/40-basic-permissions.rules
Ubuntu 9.04 em diante: /lib/udev/rules.d/50-udev-default.rules
openSUSE 11.2 em diante: /lib/udev/rules.d/50-udev-default.rules
```

Caso a distribuição seja Debian, adicione as linhas:

```
SUBSYSTEM=="usb_device", MODE="0666"
```

```
SUBSYSTEM=="usb", ENV{DEVTYPE}=="usb_device", MODE="0666"
```

ao final do arquivo:

```
Debian 6.0 em diante: /lib/udev/rules.d/91-permissions.rules
```

Para instruções de como habilitar o *hardkey* em outras distribuições Linux, entre em contato com o contato@pumatronix.com.br.

3.2.4. Variáveis de ambiente

3.2.4.1 Configuração do LD_LIBRARY_PATH

Esta configuração só é necessária para a versão Linux do SDK. Para a versão Windows, faça uma cópia da DLL para a pasta onde está o executável ou para a pasta system32

Para que a aplicação seja capaz de encontrar as bibliotecas do **Classifier** deve-se adicionar ao PATH o caminho do diretório lib do SDK. Em ambiente Linux isto pode ser feito através da variável de ambiente `LD_LIBRARY_PATH`, como exemplificado abaixo (substitua pelo caminho absoluto da instalação do SDK).

```
export LD_LIBRARY_PATH=<SDK-ROOT-DIR>/lib
```

3.3. Estrutura do SDK

O **Classifier** não foi projetado como uma aplicação *standalone* e sim como uma biblioteca a ser integrada em outras aplicações. Dessa forma, o SDK do **Classifier** é distribuído como um conjunto de *shared libraries* (.so e .dll) e seus *headers* em linguagem C.

O SDK acompanha ainda uma aplicação de exemplo (código fonte e binário pré-compilado) que pode ser utilizada como base na implementação de uma aplicação que utiliza o **Classifier**. A seção **Exemplo Básico** deste manual traz também um passo-a-passo de como implementar uma aplicação.

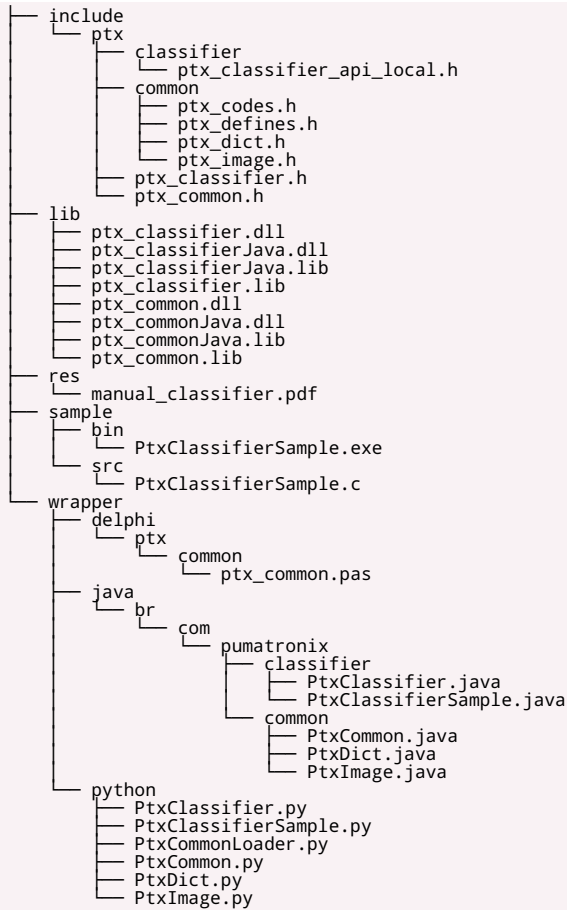
3.3.1. Árvore de arquivos versão Linux

```

include
├── ptx
│   ├── classifier
│   │   ├── ptx_classifier_api_local.h
│   │   └── common
│   │       ├── ptx_codes.h
│   │       ├── ptx_defines.h
│   │       ├── ptx_dict.h
│   │       ├── ptx_image.h
│   │       └── ptx_classifier.h
│   └── ptx_common.h
├── lib
│   ├── libptx_classifierJava.so
│   ├── libptx_classifier.so
│   ├── libptx_commonJava.so
│   └── libptx_common.so
├── sample
│   ├── bin
│   │   ├── libptx_classifierJava.so
│   │   ├── libptx_classifier.so
│   │   ├── libptx_commonJava.so
│   │   ├── libptx_common.so
│   │   └── PtxClassifierSample
│   └── src
│       └── PtxClassifierSample.c
├── wrapper
│   ├── java
│   │   └── br
│   │       └── com
│   │           └── pumatronix
│   │               ├── classifier
│   │               │   ├── PtxClassifier.java
│   │               │   └── PtxClassifierSample.java
│   │               └── common
│   │                   ├── PtxCommon.java
│   │                   ├── PtxDict.java
│   │                   └── PtxImage.java
│   └── python
│       ├── PtxClassifier.py
│       ├── PtxClassifierSample.py
│       ├── PtxCommonLoader.py
│       ├── PtxCommon.py
│       ├── PtxDict.py
│       └── PtxImage.py

```

3.3.2. Árvore de arquivos versão Windows



3.4. Arquitetura de software

O **Classifier** possui uma arquitetura simples. Todas as chamadas da API são síncronas, ou seja, bloqueiam até que o processamento esteja concluído. A função principal é a `ptx_classifier_classify`, que recebe uma imagem previamente carregada e um handle contendo a configuração e, ao final do processamento, preenche uma estrutura de resultados.

A configuração da biblioteca consiste em: tipo de cena (panorâmico ou fechado), e a mínima probabilidade para que uma detecção seja considerada um veículo válido.

Os resultados consistem em um conjunto de veículos detectados, cada um com os seguintes dados associados: classe do veículo, confiabilidade da classificação na forma de probabilidade, e coordenadas na imagem do retângulo que contém o veículo.

3.5. Exemplo de uso da API

Para um exemplo completo, veja o arquivo `sample/src/PtxClassifierSample.c` do SDK.

Para compilar a aplicação exemplo com o gcc e executar, rode a seguinte sequência de comandos a partir de um terminal no Linux:

```
gcc sample/src/PtxClassifierSample.c -I include/ -L lib/ -l ptx_classifier -l ptx_common -o PtxClassifierSample
LD_LIBRARY_PATH=lib/ ./PtxClassifierSample
```

Assim que a aplicação for iniciada, deverá aparecer na saída do terminal:

```
[PUMA] JidoshaClassifier - Client Sample
[PUMA] JidoshaClassifier library Version: X.Y.Z
[PUMA] JidoshaClassifier library SHA1: 0123456789abcdef0123456789abcdef01234567
[PUMA] ptx_common library SHA1: 123456789abcdef0123456789abcdef012345678
[PUMA] LicenseInfo - Serial: 123456789 - maxThreads: 6 - maxConnections: 0
[PUMA] LicenseInfo - State: 0 - TTL: -1 - Customer: Pumatronix
[PUMA] ./PtxClassifierSample <SCENE_TYPE> <IMG> [<IMGS>]
```

4. APIs de usuário

A biblioteca **Classifier** exporta uma API para o reconhecimento automático de características de veículos.

Por padrão, as linguagens suportadas pela API que acompanham o SDK são C, C++ e Java. Wrappers em Python, C# e Delphi podem ser fornecidos sob demanda. Em caso de dúvidas ou suporte a outras linguagens, envie um email para contato@pumatronix.com.br.

Para padronizar estruturas comuns utilizadas por seus produtos, a **Pumatronix** criou a biblioteca **ptx_common**, que implementa em C estruturas de dados, códigos de erro e outras definições comumente utilizadas. A biblioteca **ptx_common** é necessária para uso do **Classifier** e está inclusa no SDK. A documentação de sua API pode ser encontrada na seção [4.2. API ptx_common C/C++](#).

4.1. API Classifier C/C++

A API (Application Programming Interface) nativa da biblioteca está escrita em linguagem C, o que facilita a criação de bindings para uso em outras linguagens. Toda a API C está disponível através de um conjunto de headers dentro da pasta **include** do SDK.

4.1.1. API Classifier C/C++

A API contém os tipos, definições e funções para o processamento das imagens. Ela está definida no arquivo [ptx_classifier_api_local.h](#).

ptx_classifier_api_local.h

```
#include "ptx/common/ptx_defines.h"
#include "ptx/common/ptx_codes.h"
#include "ptx/common/ptx_image.h"
#include "ptx/common/ptx_dict.h"

//=====
// PtxClassifierHandle
//=====
typedef struct PtxClassifierHandle PtxClassifierHandle;

typedef enum PtxClassifierConfigs {
    PTX_CLASSIFIER_CONFIG_SCENE_TYPE           = 0,
    PTX_CLASSIFIER_CONFIG_VEHICLE_TYPE_MIN_PROB = 1,
    PTX_CLASSIFIER_CONFIG_ENUM_MAX
} PtxClassifierConfigs;

typedef enum PtxClassifierSceneType {
    PTX_CLASSIFIER_SCENE_TYPE_CLOSEUP           = 0, // Default scene
    PTX_CLASSIFIER_SCENE_TYPE_PANORAMIC        = 1,
    PTX_CLASSIFIER_SCENE_TYPE_ENUM_MAX
} PtxClassifierSceneType;

//=====
// Results
//=====

// Prediction types
typedef enum PtxClassifierResultType {
    PTX_CLASSIFIER_GET_INT_RESULTS_SIZE           = 0,
    PTX_CLASSIFIER_GET_PTxDICT_AT_RESULT          = 1,
    PTX_CLASSIFIER_GET_INT_VEHICLE_TYPE_CLASS    = 2,
    PTX_CLASSIFIER_GET_FLOAT_VEHICLE_TYPE_PROB   = 3,
    PTX_CLASSIFIER_GET_INT_VEHICLE_X_POINTS_SIZE = 5,
    PTX_CLASSIFIER_GET_INT_AT_VEHICLE_X_POINT   = 4,
    PTX_CLASSIFIER_GET_INT_VEHICLE_Y_POINTS_SIZE = 7,
    PTX_CLASSIFIER_GET_INT_AT_VEHICLE_Y_POINT   = 6,
    PTX_CLASSIFIER_GET_ENUM_MAX
} PtxClassifierResultType;

typedef enum PtxClassifierVehicleType {
    PTX_CLASSIFIER_VEHICLE_TYPE_UNKNOWN          = 0,
    PTX_CLASSIFIER_VEHICLE_TYPE_CAR              = 1,
    PTX_CLASSIFIER_VEHICLE_TYPE_MOTORCYCLE      = 2,
    PTX_CLASSIFIER_VEHICLE_TYPE_TRUCK            = 3,
    PTX_CLASSIFIER_VEHICLE_TYPE_BUS              = 4,
    PTX_CLASSIFIER_VEHICLE_TYPE_ENUM_MAX
} PtxClassifierVehicleType;

//=====
// FUNCTIONS
//=====

/* Classifier functions */
PTXEXPORT int PTXAPI ptx_classifier_init();
PTXEXPORT int PTXAPI ptx_classifier_destroy();
PTXEXPORT int PTXAPI ptx_classifier_get_version(int* major, int* minor, int* release);
PTXEXPORT const char* PTXAPI ptx_classifier_get_SHA1();

/* Classifier handle */
PTXEXPORT PtxClassifierHandle* PTXAPI ptx_classifier_create_handle();
PTXEXPORT int PTXAPI ptx_classifier_free_handle(PtxClassifierHandle* handle);
PTXEXPORT int PTXAPI ptx_classifier_set_config(PtxClassifierHandle* handle, PtxDict* config);

/* Classifier processing */
PTXEXPORT int PTXAPI ptx_classifier_classify(PtxClassifierHandle* handle, PtxImage* img, PtxDict* results);

/* License */
PTXEXPORT int PTXAPI ptx_classifier_get_license_info(PtxProductLicenseInfo* license);
```

4.1.1.1. Tipos

enum PtxClassifierConfigs

Descrição

Define as configurações disponíveis da biblioteca. As configurações são alteradas através da função [ptx_classifier_set_config](#).

Membros

`PTX_CLASSIFIER_CONFIG_SCENE_TYPE`: tipo de configuração de cena a ser usado para classificação.

`PTX_CLASSIFIER_CONFIG_VEHICLE_TYPE_MIN_PROB`: configuração da probabilidade mínima admitida de retorno por parte do **Classifier**. Classificações com probabilidade menor que essa serão descartadas internamente pela biblioteca e não serão retornadas ao usuário.

enum PtxClassifierSceneType

Descrição

Define as cenas que podem ser configuradas no **Classifier**, permitindo um melhor funcionamento da biblioteca dependendo da instalação.

Membros

`PTX_CLASSIFIER_SCENE_TYPE_CLOSEUP`: Tipo de cena em que o veículo ocupa *mais* de 80% da imagem.

`PTX_CLASSIFIER_SCENE_TYPE_PANORAMIC`: Tipo de cena onde os veículos ocupam *menos* de 60% da imagem.

Nota: Para veículos ocupando entre 60% e 80% da imagem é interessante experimentar cada um dos tipos de cena e verificar qual delas apresenta melhor desempenho.

enum PtxClassifierResultType

Descrição

Chaves para requisitar do retorno do **Classifier** (tipo `ptx_dict`) os valores das propriedades dos veículos.

Membros

`PTX_CLASSIFIER_GET_INT_RESULTS_SIZE`: Chave para requisitar o valor `int` da quantidade de resultados retornados.

`PTX_CLASSIFIER_GET_PTXDICT_AT_RESULT`: Chave para requisitar o valor `ptx_dict` de uma determinada posição contendo um único resultado.

`PTX_CLASSIFIER_GET_INT_VEHICLE_TYPE_CLASS`: Chave para requisitar o valor `int` que representa o tipo de veículo encontrado.

`PTX_CLASSIFIER_GET_FLOAT_VEHICLE_TYPE_PROB`: Chave para requisitar o valor `float` que representa a confiabilidade em forma de probabilidade para o tipo de veículo.

`PTX_CLASSIFIER_GET_INT_VEHICLE_X_POINTS_SIZE`: Chave para requisitar o valor `int` que representa quantos valores x são utilizados para descrever o contorno do objeto.

`PTX_CLASSIFIER_GET_INT_AT_VEHICLE_X_POINT`: Chave para requisitar o valor `int` que corresponde a um único valor x pertencente ao contorno do objeto.

`PTX_CLASSIFIER_GET_INT_VEHICLE_Y_POINTS_SIZE`: Chave para requisitar o valor `int` que representa quantos valores y são utilizados para descrever o contorno do objeto.

`PTX_CLASSIFIER_GET_INT_AT_VEHICLE_Y_POINT`: Chave para requisitar o valor `int` que corresponde a um único valor y pertencente ao contorno do objeto.

enum PtxClassifierVehicleType

Descrição

Campos representando os possíveis retornos da chave `PTX_CLASSIFIER_GET_INT_VEHICLE_TYPE_CLASS`, ou seja, os possíveis tipos de veículos retornados pela biblioteca.

Membros

`PTX_CLASSIFIER_VEHICLE_TYPE_UNKNOWN`: Tipo de veículo que não conseguiu ser categorizado

`PTX_CLASSIFIER_VEHICLE_TYPE_CAR`: Veículo do tipo carro

`PTX_CLASSIFIER_VEHICLE_TYPE_MOTORCYCLE`: Veículo do tipo moto

`PTX_CLASSIFIER_VEHICLE_TYPE_TRUCK`: Veículo do tipo caminhão

`PTX_CLASSIFIER_VEHICLE_TYPE_BUS`: Veículo do tipo ônibus

`struct PtxClassifierHandle`

Descrição

Estrutura que contém as configurações internas e estados da biblioteca.

4.1.1.2. Métodos

`ptx_classifier_init`

Protótipo da Função

```
int ptx_classifier_init();
```

Descrição

Função utilizada para inicializar a biblioteca após sua carga.

Parâmetros

Nenhum

Retorno

Um inteiro com código de retorno de função.

`ptx_classifier_destroy`

Protótipo da Função

```
int ptx_classifier_destroy();
```

Descrição

Função utilizada para descarregar a biblioteca.

Parâmetros

Nenhum

Retorno

Um inteiro com código de retorno de função.

`ptx_classifier_get_version`

Protótipo da Função

```
int ptx_classifier_get_version(int* major, int* minor, int* release)
```

Descrição

Função utilizada para consultar a versão da biblioteca. O **Classifier** segue um sistema de versionamento semântico **major.minor.release**. O número **major** é incrementado quando a versão tem quebra de API em relação à versão anterior. Caso não tenha quebra de API e haja inclusão de nova funcionalidade, o número **minor** é incrementado. Caso não haja quebra de API nem inclusão de nova funcionalidade, o número **release** é incrementado (é o caso de correções de bugs e alterações pequenas).

Parâmetros

`int *major, int *minor, int *release`: ponteiros para variáveis inteiras onde serão escritos os números que compõem a versão.

Retorno

Um inteiro com código de retorno de função.

ptx_classifier_get_SHA1

Protótipo da Função

```
const char* ptx_classifier_get_SHA1();
```

Descrição

Função utilizada para verificar o hash SHA1 do build da biblioteca. Essa string é utilizada para **Pumatronix** para rastreamento do build.

Parâmetros

Nenhum

Retorno

Retorna um ponteiro para o início de uma string terminada em `\0` contendo o hash SHA1 do build.

ptx_classifier_create_handle

Protótipo da Função

```
PtxClassifierHandle* ptx_classifier_create_handle();
```

Descrição

Função utilizada para alocar a memória para a configuração da biblioteca. No caso de uso multithread, cada thread deverá chamar `ptx_classifier_create_handle` e usar seu próprio `PtxClassifierHandle`. O mesmo `PtxClassifierHandle` pode ser utilizado quantas vezes for necessário, desde que de apenas uma thread por vez.

As configurações da biblioteca, feitas através da função `ptx_classifier_set_config`, são armazenadas no `PtxClassifierHandle`. Portanto, cada `PtxClassifierHandle` pode ter uma configuração diferente. Isso pode ser útil, por exemplo, quando deseja-se utilizar a biblioteca **Classifier** para processar imagens oriundas de diversas câmeras, e deseja-se aplicar configurações diferentes para imagens de câmeras diferentes.

Parâmetros

Nenhum

Retorno

Retorna um ponteiro para um struct tipo `PtxClassifierHandle` que será utilizado nas chamadas das funções subsequentes.

ptx_classifier_free_handle

Protótipo da Função

```
int ptx_classifier_free_handle(PtxClassifierHandle* handle);
```

Descrição

Função utilizada para liberar a memória alocada para o objeto do tipo `PtxClassifierHandle`.

Parâmetros

`PtxClassifierHandle* handle`: Ponteiro para o handle do tipo `PtxClassifierHandle`.

Retorno

Um inteiro com código de retorno de função.

ptx_classifier_set_config

Protótipo da Função

```
int ptx_classifier_set_config(PtxClassifierHandle* handle, PtxDict* config);
```

Descrição

Função utilizada para aplicar a configuração feita através de um campo `PtxDict`.

Parâmetros

`PtxClassifierHandle* handle`: Ponteiro para o handle do tipo `PtxClassifierHandle`

`PtxDict* config`: Ponteiro para objeto do tipo `PtxDict`

Retorno

Um inteiro com código de retorno de função.

`ptx_classifier_classify`

Protótipo da Função

```
int ptx_classifier_classify(PtxClassifierHandle* handle, PtxImage* img, PtxDict* results);
```

Descrição

Função utilizada para processar e extrair as informações da imagem e retorná-las através do parâmetro `results`.

Nota: Esta função pode ser bastante demorada, levando centenas de milisegundos para terminar ou até mais, dependendo da CPU utilizada.

Parâmetros

`PtxClassifierHandle* handle`: Ponteiro para o handle do tipo `PtxClassifierHandle`

`PtxImage* img`: Ponteiro para imagem do tipo `PtxImage`

`PtxDict* results`: Ponteiro para resultados do tipo `PtxDict`

Retorno

Um inteiro com código de retorno de função.

`ptx_classifier_get_license_info`

Protótipo da Função

```
int ptx_classifier_get_license_info(PtxProductLicenseInfo* license);
```

Descrição

Função utilizada para requisitar as informações de licença em relação ao produto **Classifier**.

Parâmetros

`PtxProductLicenseInfo* license`: Ponteiro para o objeto do tipo `PtxProductLicenseInfo`

Retorno

Um inteiro com código de retorno de função.

4.2. API `ptx_common` C/C++

A biblioteca `ptx_common` implementa em C estruturas de dados, códigos de erro e outras definições comumente utilizadas por produtos da **Pumatronix**. Sua API está definida no arquivo `ptx_common.h`, que por si inclui outros headers.

4.2.1. `ptx_codes.h`

```
//=====
// RETURN CODES
//=====
enum PtxCommonReturnCode {
    /* API CALL RETURN CODES */
    /* SUCCESS */
    PTX_SUCCESS = 0,

    /* BASIC ERRORS */
    PTX_FILE_NOT_FOUND = 1,
    PTX_INVALID_IMAGE = 2,
    PTX_INVALID_IMAGE_TYPE = 3,
    PTX_INVALID_PARAMETER = 4,
    PTX_COUNTRY_NOT_SUPPORTED = 5,
    PTX_API_CALL_NOT_SUPPORTED = 6,
    PTX_INVALID_ROI = 7,
    PTX_INVALID_HANDLE = 8,
    PTX_API_CALL_HAS_NO_EFFECT = 9,
    PTX_INVALID_IMAGE_SIZE = 10,

    /* LICENSE ERRORS */
    PTX_LICENSE_INVALID = 16,
    PTX_LICENSE_EXPIRED = 17,
    PTX_LICENSE_MAX_THREADS_EXCEEDED = 18,
    PTX_LICENSE_UNTRUSTED_RTC = 19,
    PTX_LICENSE_MAX_CONNS_EXCEEDED = 20,
    PTX_LICENSE_UNAUTHORIZED_PRODUCT = 21,

    /* NETWORK ERRORS */
    PTX_CONNECT_FAILED = 100,
    PTX_SOCKET_DISCONNECT = 101,
    PTX_SOCKET_QUEUE_TIMEOUT = 202,
    PTX_SOCKET_QUEUE_FULL = 103,
    PTX_SOCKET_IO_ERROR = 104,
    PTX_SOCKET_WRITE_FAILED = 105,
    PTX_SOCKET_READ_TIMEOUT = 106,
    PTX_INVALID_RESPONSE = 107,
    PTX_HANDLE_QUEUE_FULL = 108,
    PTX_INVALID_REQUEST = 109,
    PTX_INVALID_MESSAGE = 110,
    PTX_INVALID_STREAM_FRAME = 209,
    PTX_FRAME_QUEUE_FULL = 211,
    PTX_LAST_FRAME_UNAVAILABLE = 212,
    PTX_SERVER_CONN_LIMIT_REACHED = 213,
    PTX_SERVER_VERSION_NOT_SUPPORTED = 214,

    /* MJPEG ERRORS */
    PTX_MJPEG_ERROR_BASE = 1000,
    PTX_MJPEG_HTTP_HEADER_OVERFLOW = 1001,
    PTX_MJPEG_HTTP_RESPONSE_NOT_OK = 1002,
    PTX_MJPEG_HTTP_CONTENT_TYPE_ERROR = 1003,
    PTX_MJPEG_HTTP_CONTENT_LENGTH_ERROR = 1004,
    PTX_MJPEG_HTTP_FRAME_BOUNDARY_NOT_FOUND = 1005,
    PTX_MJPEG_CONNECTION_CLOSED = 1006,
    PTX_MJPEG_CONNECT_FAILED = 1007,

    /* HARDWARE ERRORS - returned by the process */
    PTX_HW_FPGA_INIT_FAILED = 2000,
    PTX_HW_FPGA_LOCK_FAILED = 2001,

    /* OTHERS */
    PTX_UNKNOWN_ERROR = 99999
};

typedef struct PtxProductLicenseInfo
{
    uint64_t serial;
    char customer[64];
    int maxThreads;
    int maxConnections;
    int state;
    int ttl;
} PtxProductLicenseInfo;

PTXEXPORT const char* PTXAPI ptx_common_get_SHA1();
```

4.2.1.1. Tipos

`enum PtxCommonReturnCode`

Descrição

Define os códigos de erro da biblioteca `ptx_common`.

Membros

Os vários membros desta enumeração são retornados por funções em diversos tipos de situações: sucesso, imagem inválida, parâmetro inválido, erros de licença, erros de rede etc.

enum PtxProductLicenseInfo

Descrição

Struct utilizada para armazenar as informações sobre a licença utilizada pela biblioteca.

Membros

- `uint64_t serial` : serial number da licença
- `char customer[64]` : nome do cliente que adquiriu a licença
- `int maxThreads` : número máximo de threads de processamento habilitadas
- `int maxConnections` : número máximo de conexões paralelas habilitadas
- `int state` : estado da licença - ver [PtxCommonReturnCode](#)
- `int ttl` : time-to-live em horas para licenças do tipo RTC. Este campo possui o valor `-1` caso a licença não seja expirável

4.2.1.2. Métodos

ptx_common_get_SHA1

Protótipo da Função

```
const char* PTXAPI ptx_common_get_SHA1();
```

Descrição

Função utilizada para verificar o hash SHA1 do build da biblioteca **ptx_common**. Essa string é utilizada para **Pumatronix** para rastreamento do build.

Parâmetros

Nenhum

Retorno

Retorna um ponteiro para o início de uma string terminada em `\0` contendo o hash SHA1 do build.

4.2.2. ptx_dict.h

Este arquivo define uma estrutura de dados de dicionário, usada para armazenar relações do tipo chave-valor. A chave é sempre do tipo `int`. Cada produto da **Pumatronix** que utiliza a biblioteca `ptx_common` enumera em seu header quais chaves utiliza.

```
//=====
// PtxDict
//=====
typedef struct PtxDict PtxDict;

/* lifecycle */
PTXEXPORT PtxDict* PTXAPI ptxdict_create();
PTXEXPORT int PTXAPI ptxdict_free(PtxDict* ptx_dict);

/* Setters */
PTXEXPORT int PTXAPI ptxdict_set_int(PtxDict* ptx_dict, int key, int value);
PTXEXPORT int PTXAPI ptxdict_set_float(PtxDict* ptx_dict, int key, float value);

/* Getters */
PTXEXPORT int PTXAPI ptxdict_get_int(PtxDict* ptx_dict, int key, int* value);
PTXEXPORT int PTXAPI ptxdict_get_float(PtxDict* ptx_dict, int key, float* value);
PTXEXPORT int PTXAPI ptxdict_get_int_at(PtxDict* ptx_dict, int key, int index, int* value);
PTXEXPORT int PTXAPI ptxdict_get_ptxdict_at(PtxDict* ptx_dict, int key, int index, PtxDict* value);
```

4.2.2.1. Tipos

struct PtxDict

Descrição

Struct opaca utilizada para armazenar o dicionário.

Membros

Nenhum, por ser uma estrutura opaca.

4.2.2.2. Métodos

ptxdict_create

Protótipo da Função

```
PtxDict* PTXAPI ptxdict_create();
```

Descrição

Função utilizada para alocar memória para um dicionário.

Parâmetros

Nenhum

Retorno

Retorna um ponteiro para um struct tipo `[PtxDict](#struct-ptxdict)` que será utilizado nas chamadas das funções subsequentes.

ptxdict_free

Protótipo da Função

```
int ptxdict_free(PtxDict* ptx_dict);
```

Descrição

Função utilizada para liberar a memória alocada para o objeto do tipo `PtxDict`.

Parâmetros

`PtxDict* dict`: Pontoeiro para objeto do tipo `PtxDict` cuja memória será liberada.

Retorno

Um inteiro com código de retorno de função.

ptxdict_set_int

Protótipo da Função

```
int ptxdict_set_int(PtxDict* ptx_dict, int key, int value);
```

Descrição

Função utilizada para armazenar um par chave-valor.

Parâmetros

PtxDict* dict: Ponteiro para objeto do tipo **PtxDict** onde o par chave-valor será armazenado.

int key: chave do tipo int.

int value: valor do tipo int que será armazenado como associado à **key**.

Retorno

Um inteiro com código de retorno de função.

ptxdict_set_float

Protótipo da Função

```
int ptxdict_set_float(PtxDict* ptx_dict, int key, float value);
```

Descrição

Função utilizada para armazenar um par chave-valor.

Parâmetros

PtxDict* dict: Ponteiro para objeto do tipo **PtxDict** onde o par chave-valor será armazenado.

int key: chave do tipo int.

float value: valor do tipo float que será armazenado como associado à **key**.

Retorno

Um inteiro com código de retorno de função.

ptxdict_get_int

Protótipo da Função

```
int ptxdict_get_int(PtxDict* ptx_dict, int key, int* value);
```

Descrição

Função utilizada para requisitar um valor associado a uma chave.

Parâmetros

PtxDict* dict: Ponteiro para objeto do tipo **PtxDict**.

int key: chave do tipo int cujo valor deseja-se requisitar.

int* value: ponteiro para int onde será escrito o valor associado à **key**, caso exista.

Retorno

Um inteiro com código de retorno de função.

ptxdict_get_float

Protótipo da Função

```
int ptxdict_get_float(PtxDict* ptx_dict, int key, float* value);
```

Descrição

Função utilizada para requisitar um valor associado a uma chave.

Parâmetros

PtxDict* dict: Ponteiro para objeto do tipo **PtxDict**.

int key: chave do tipo int cujo valor deseja-se requisitar.

float* value: ponteiro para float onde será escrito o valor associado à **key**, caso exista.

Retorno

Um inteiro com código de retorno de função.

ptxdict_get_int_at

Protótipo da Função

```
int ptxdict_get_int_at(PtxDict* ptx_dict, int key, int index, int* value);
```

Descrição

Função utilizada para requisitar o valor em um determinado índice de um vetor associado a uma chave.

Parâmetros

PtxDict* dict: Ponteiro para objeto do tipo **PtxDict**.

int key: chave do tipo int cujo valor deseja-se requisitar.

int index: índice desejado do vetor associado à **key**.

int* value: ponteiro para float onde será escrito o valor, caso exista.

Retorno

Um inteiro com código de retorno de função.

ptxdict_get_ptxdict_at

Protótipo da Função

```
int ptxdict_get_ptxdict_at(PtxDict* ptx_dict, int key, int index, PtxDict* value);
```

Descrição

Função utilizada para requisitar o valor em um determinado índice de um vetor associado a uma chave.

Parâmetros

PtxDict* dict: Ponteiro para objeto do tipo **PtxDict**.

int key: chave do tipo int cujo valor deseja-se requisitar.

int index: índice desejado do vetor associado à **key**.

PtxDict* value: ponteiro para **PtxDict** onde será escrito o valor, caso exista.

Retorno

Um inteiro com código de retorno de função.

4.2.3. ptx_image.h

Este arquivo define uma estrutura para carregar imagens. Suporta tanto imagens estruturadas (jpg e bmp) quanto imagens RAW.

```
//=====
// Types
//=====

typedef struct PtxImage PtxImage;

/* Raw image pixel format */
typedef enum PtxImagePixelFormat {
    PTX_IMG_FMT_XRGB_8888      = 0,
    PTX_IMG_FMT_RGB_888       = 1,
    PTX_IMG_FMT_LUMA          = 2,
    PTX_IMG_FMT_YUV420        = 3,
    PTX_IMG_FMT_YUV_NV12      = 4,
} PtxImagePixelFormat;

//=====
// Functions
//=====

/* lifecycle */
PTXEXPORT PtxImage* PTXAPI ptximage_create();
PTXEXPORT int PTXAPI ptximage_free(PtxImage* img);

/* load */
PTXEXPORT int PTXAPI ptximage_load_from_file(PtxImage* img, const char* filename);
PTXEXPORT int PTXAPI ptximage_load_from_memory(PtxImage* img, const uint8_t* buffer, int bufferSize);
PTXEXPORT int PTXAPI ptximage_load_from_raw_format(PtxImage* img, const uint8_t* buffer, int width, int height, int stride,
```

4.2.3.1. Tipos

struct PtxImage

Descrição

Struct opaca utilizada para carregar imagens.

Membros

Nenhum, por ser uma estrutura opaca.

enum PtxImagePixelFormat

Descrição

Enumeração dos tipos de imagem RAW que podem ser carregados.

Membros

Nenhum, por ser uma estrutura opaca.

4.2.3.2. Métodos

ptximage_create

Protótipo da Função

```
PtxImage* ptximage_create();
```

Descrição

Função utilizada para alocar memória para uma estrutura de imagem.

Parâmetros

Nenhum

Retorno

Retorna um ponteiro para um struct tipo [PtxImage](#struct-ptximage) que será utilizado nas chamadas das funções subsequentes.

ptximage_free

Protótipo da Função

```
int ptximage_free(PtxImage* img);
```

Descrição

Função utilizada para liberar a memória alocada para um objeto do tipo **PtxImage**.

Parâmetros

PtxImage* img: Ponteiro para objeto do tipo **PtxImage** cuja memória será liberada.

Retorno

Um inteiro com código de retorno de função.

ptximage_load_from_file

Protótipo da Função

```
int ptximage_load_from_file(PtxImage* img, const char* filename);
```

Descrição

Função utilizada para carregar uma imagem estruturada (jpg ou bmp) a partir de um arquivo.

Parâmetros

PtxImage* img: Ponteiro para objeto do tipo **PtxImage** onde será armazenada a imagem em memória.

const char* filename: Arquivo que contém uma imagem estruturada a ser carregada.

Retorno

Um inteiro com código de retorno de função.

ptximage_load_from_memory

Protótipo da Função

```
int ptximage_load_from_memory(PtxImage* img, const uint8_t* buffer, int bufferSize);
```

Descrição

Função utilizada para carregar uma imagem estruturada (jpg ou bmp) a partir de um buffer em memória.

Parâmetros

PtxImage* img: Ponteiro para objeto do tipo **PtxImage** onde será armazenada a imagem em memória.

const uint8* buffer: Buffer contendo a imagem estruturada a ser carregada.

int bufferSize: Tamanho de **buffer** em bytes.

Retorno

Um inteiro com código de retorno de função.

ptximage_load_from_raw_format

Protótipo da Função

```
int ptximage_load_from_raw_format(PtxImage* img, const uint8_t* buffer, int width, int height, int stride, PtxImagePixelFormat)
```

Descrição

Função utilizada para carregar uma imagem RAW a partir de um buffer em memória. Os tipos de imagens suportados estão definidos por [PtImagePixelFormat](#).

Parâmetros

`PtImage* img`: Ponteiro para objeto do tipo [PtImage](#) onde será armazenada a imagem em memória.

`const uint8* buffer`: Buffer contendo a imagem RAW a ser carregada.

`int width`: Largura da imagem em pixels.

`int height`: Altura da imagem em pixels.

`int stride`: Número de bytes entre uma linha e a linha seguinte da imagem.

`PtImagePixelFormat fmt`: Tipo de imagem RAW, conforme [PtImagePixelFormat](#).

Retorno

Um inteiro com código de retorno de função.

Índice da API

C/C++

- [enum PtxClassifierConfigs](#)
- [enum PtxClassifierResultType](#)
- [enum PtxClassifierSceneType](#)
- [enum PtxClassifierVehicleType](#)
- [enum PtxCommonReturnCode](#)
- [enum PtxImagePixelFormat](#)
- [enum PtxProductLicenseInfo](#)
- [ptx_classifier_classify](#)
- [ptx_classifier_create_handle](#)
- [ptx_classifier_destroy](#)
- [ptx_classifier_free_handle](#)
- [ptx_classifier_get_license_info](#)
- [ptx_classifier_get_SHA1](#)
- [ptx_classifier_get_version](#)
- [ptx_classifier_init](#)
- [ptx_classifier_set_config](#)
- [ptx_common_get_SHA1](#)
- [ptxdict_create](#)
- [ptxdict_free](#)
- [ptxdict_get_float](#)
- [ptxdict_get_int](#)
- [ptxdict_get_int_at](#)
- [ptxdict_get_ptxdict_at](#)
- [ptxdict_set_float](#)
- [ptxdict_set_int](#)
- [ptximage_create](#)
- [ptximage_free](#)
- [ptximage_load_from_file](#)
- [ptximage_load_from_memory](#)
- [ptximage_load_from_raw_format](#)
- [struct PtxClassifierHandle](#)
- [struct PtxDict](#)
- [struct PtxImage](#)