JIDOSHALIGHT
Manual de Integração

Revisão 1.0

PUMATRONIX

www.pumatronix.com

Pumatronix Equipamentos Eletrônicos Ltda.

Rua Bartolomeu Lourenço de Gusmão, 1970. Curitiba, Brasil

Copyright 2020 Pumatronix Equipamentos Eletrônicos Ltda.

Todos os direitos reservados.

Visite nosso website http://www.pumatronix.com

Envie comentários sobre este documento no e-mail suporte@pumatronix.com

Informações contidas neste documento estão sujeitas a mudança sem aviso prévio.

A Pumatronix se reserva o direito de modificar ou melhorar este material sem obrigação de notificação das alterações ou melhorias.

A Pumatronix assegura permissão para download e impressão deste documento, desde que a cópia eletrônica ou física deste documento contenha o texto na íntegra. Qualquer alteração neste conteúdo é estritamente proibida.

Histórico de Alterações

Data	Revisão	Conteúdo atualizado
22/12/2022	1.0	Versão Inicial

Visão Geral

Este documento tem o objetivo de orientar o desenvolvedor na aplicação da biblioteca de software *JidoshaLight* responsável pelo reconhecimento e leitura de placas veiculares (LPR) a partir da análise de imagens e aplicável em softwares compatíveis com a biblioteca. Neste documento estão detalhadas as opções de configuração do kit de desenvolvimento de software (SDK) e das APIs disponibilizadas.

De acordo com a versão da biblioteca aplicada ao software, alguns formatos de placas veiculares podem não ser suportadas e algumas funções podem ser disponibilizadas somente nas versões mais atuais.

Sumário

1 Estrutura do SDK JidoshaLight 6
1.1 APIs Suportadas 6
2 JidoshaLight Linux 7
2.1 Arquitetura de software do JidoshaLight Linux 7
2.1.1 Restrições JidoshaLight Linux 9
2.2 Instalação do JidoshaLight Linux 9
2.2.1 Configuração das permissões do *hardkey* 9
2.2.2 Configuração das variáveis de ambiente 9
2.2.3 Configuração do arquivo de preferências 11
2.3 Aplicações exemplo 12
3 JidoshaLight Windows 13
3.1 Instalação do JidoshaLight Windows 13
3.1.1 Configuração das variáveis de ambiente 14
3.1.2 Configuração do arquivo de preferências 14
3.2 Aplicações exemplo 14
4 JidoshaLight Linux/FPGA 15
4.1 Arquitetura de software do JidoshaLight Linux/FPGA 15
4.1.1 Restrições JidoshaLight Linux/FPGA 16
4.2 Instalação do JidoshaLight Linux/FPGA 17
4.2.1 Configuração da licença 17
4.2.2 Configuração das variáveis de ambiente 17
4.2.3 Configuração do kernel Linux 17
4.3 Aplicações exemplo 18
5 JidoshaLight Android™ 18
5.1 Arquitetura de software do JidoshaLight Android™ 19
5.1.1 Restrições JidoshaLight Android™ 20
5.2 Instalação do JidoshaLight Android™ 21
5.2.1 Licenciamento 21
5.2.2 Permissões 21
5.3 Aplicativo Exemplo 21

5.3.1 Package `br.gaussian.io` 22
5.3.2 Package `br.gaussian.jidoshalight` 22
5.3.3 Package `br.gaussian.jidoshalight.camera` 22
5.3.4 Package `br.gaussian.jidoshalight.sample` 22
6 APIs de usuário 26
6.1 Limitações conhecidas 26
6.2 API JidoshaLight C/C++ 26
6.2.1 API JidoshaLight C/C++ (Local) 26
6.2.2 API JidoshaLight C/C++ (Remota Síncrona) 45
6.2.3 API JidoshaLight C/C++ (Remota Assíncrona) 46
6.2.4 API JidoshaLight C/C++ (Servidor) 52
6.3 API JidoshaLight Java 54
6.3.1 API JidoshaLight Java (Local) 54
6.3.2 API JidoshaLight Java (Remota Assíncrona) 66
6.3.3 API JidoshaLight Java (Servidor) 70
6.3.4 API JidoshaLight Java (IO/Mjpeg) 72
6.4 Guia de Migração - API 1 C/C++ JIDOSHA 75
7 APIs de usuário do JIDOSHA 76
7.1 jidoshaCore.h 77
7.2 API1 JIDOSHA C/C++ 80
7.2.1 Tipos 80
7.2.2 Métodos 80
7.3 API2 JIDOSHA C/C++ 82
7.3.1 Tipos 82
7.3.2 Métodos 82
7.3.3 API 2 - Configuração 87
7.3.4 API 2 - Configuração de perspectiva da imagem 92
7.4 API JIDOSHA C# / VB.NET 94
7.4.1 API 1 94
7.4.2 Exemplos API JIDOSHA C# / VB.NET 96
7.5 API JIDOSHA Delphi 98
7.5.1 API 1 98
7.5.2 Exemplo API JIDOSHA Delphi 99

7.6 API JIDOSHA Java 99
7.6.1 API 1 99
7.6.2 Exemplo API JIDOSHA Java 100
7.7 Builds especiais da API legada 100

1 Estrutura do SDK JidoshaLight

Todos os caminhos utilizados neste manual são relativos ao diretório raiz do SDK *JidoshaLight_TARGET_x.y.z*. O SDK é o kit de desenvolvimento de software do *JidoshaLight* composto por:

- bibliotecas de reconhecimento de placas libjidoshaLight.so, libjidoshaLightRemote.so, libjidoshaLightJava.so;
- respectivas APIs das bibliotecas;
- pelos wrappers (bindings) para outras linguagens;
- pelas aplicações de exemplo pré-compiladas;
- pelo código fonte destas aplicações;
- por um script básico de compilação;
- pelo Manual de Integração;
- por uma imagem de placa para testes.

A estrutura do pacote de dados do SDK contém:

Pasta	Conteúdo
res	pasta contendo os manuais e uma foto para usar de exemplo na execução do software
lib	pasta contendo as bibliotecas (.so ou .dll)
includes	pastas com os headers a serem incluídos em aplicações C
sample	pasta contendo os códigos de amostra para C *Nota: Na pasta sample/bin estão mini programas para testar o funcionamento.
wrappers	pasta contendo os bindings para outros tipos de linguagem
jidoshapc	pasta contendo bindings para a antiga interface do *Jidosha*

1.1 APIs Suportadas

A API (Interface de Programação de Aplicativos) primária do Jidosha é a API *JidoshaLight C/C++* e pode ser encontrada dentro das pastas *include* e *lib*. Wrappers (bindings) para outras linguagens são fornecidos junto com o SDK e estão dentro da pasta wrappers:

1) JidoshaLight C/C++;
2) JidoshaLight Java (1.7+);
3) JidoshaLight Android;
4) JidoshaLight Python (2.7 e 3.x);
5) JidoshaLight C#.

Para integração com outras linguagens ainda não suportadas, entre em contato com o Suporte Técnico.

O SDK também fornece um conjunto de API legadas dentro da pasta legacy. Essas APIs não recebem novas funcionalidades e existem apenas para garantir o suporte a aplicações legadas desenvolvidas a partir do *Jidosha* (versão 1.7.0 ou inferior). Internamente essa API utiliza a API *JidoshaLight C/C++* padrão e, portanto, gera os mesmos resultados de reconhecimento.

Atenção às APIS que NÃO são recomendadas para novos designs:

1) **jidoshapc C/C+**
2) **jidoshapc Java (1.6+)**
3) **jidoshapc Python (2.7)**
4) **jidoshapc Delphi (apenas Windows)**
5) **jidoshapc C#**

2 JidoshaLight Linux

A biblioteca de software *JidoshaLight Linux* foi criada para funcionar em conjunto com o *hardkey* (chave de segurança) que acompanha a biblioteca. Ou seja, para o correto funcionamento da biblioteca o referido *hardkey* deverá estar conectado à USB do ambiente em que a biblioteca será utilizada. Existem duas versões de *hardkey*, uma para uso geral e a versão de demonstração, com data de validade. Quando a data de validade expira, a biblioteca automaticamente passa a retornar placas vazias. Se seu *hardkey* de demonstração expirar e você desejar comprar uma licença ou estender o período de demonstração, entre em contato com a Pumatronix.

Verifique os pré-requisitos de instalação explicitados no Manual de Produto.

2.1 Arquitetura de software do JidoshaLight Linux

As chamadas à API da biblioteca podem ser feitas de forma local ou remota através de uma rede IP.

As chamadas locais são executadas na mesma thread em que foi realizada a chamada. Para licenças com mais de 1 thread habilitada ou para casos onde a thread principal não pode ser bloqueada enquanto a imagem é processada, deve-se criar novas threads para o processamento.

As chamadas remotas podem ser síncronas ou assíncronas. Em ambos os casos as chamadas são feitas localmente e as imagens são processadas remotamente em um servidor. A licença de uso é necessária apenas no servidor que executa o algoritmo, não sendo necessária para o uso da biblioteca remota.

As chamadas síncronas são bloqueantes e retornam o resultado do processamento ao final da chamada.

No caso da interface assíncrona, a chamada retorna imediatamente e o resultado do processamento é retornado através de uma *callback* de usuário.

A figura a seguir apresenta um diagrama com a arquitetura sugerida para uma aplicação Linux que utiliza a biblioteca JidoshaLight com chamadas locais, seja ela single thread ou multithread. Para o correto funcionamento da aplicação, a biblioteca `*libjidoshaLight.so*` deve estar linkada à aplicação e o *hardkey* deve estar plugado à máquina. Em seguida, para o caso single thread, basta chamar as funções da API. Já para o caso multithread, a aplicação deve criar as threads de processamento necessárias e, a partir destas, fazer as chamadas às funções da API da biblioteca *JidoshaLight*.

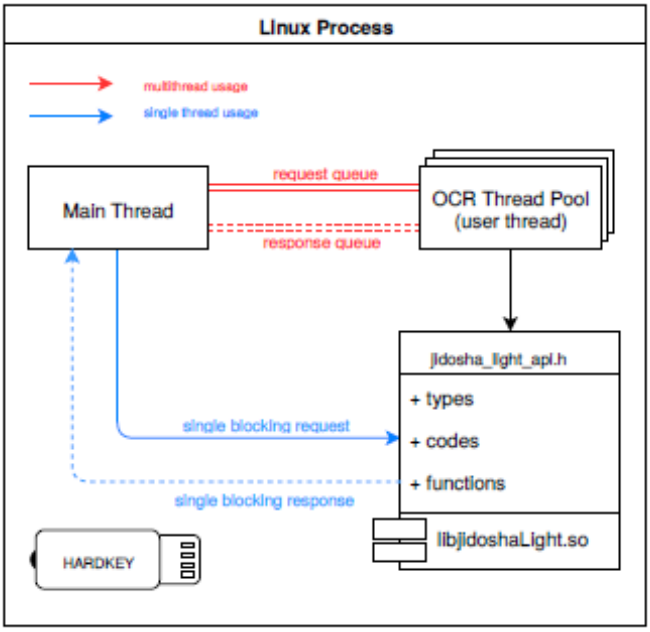


Figura 1 – Diagrama com a arquitetura sugerida para uma aplicação Linux

A figura a seguir mostra a arquitetura sugerida para o uso da biblioteca com chamadas remotas. Para o correto funcionamento da aplicação, a biblioteca `*libjidoshaLightRemote.so*` deve estar linkada à aplicação cliente. A biblioteca `*libjidoshaLight.so*` deve estar linkada à aplicação servidor e o *hardkey* deve estar plugado à máquina. As aplicações cliente e servidor devem ser interligadas através de uma rede TCP/IPv4, real ou virtual (loopback, por exemplo). Apesar de não ilustrado na figura, da mesma forma que para as chamadas locais a aplicação cliente poderá ter várias threads, sendo que o servidor poderá limitar o número de sessões ativas concorrentes em função da licença.

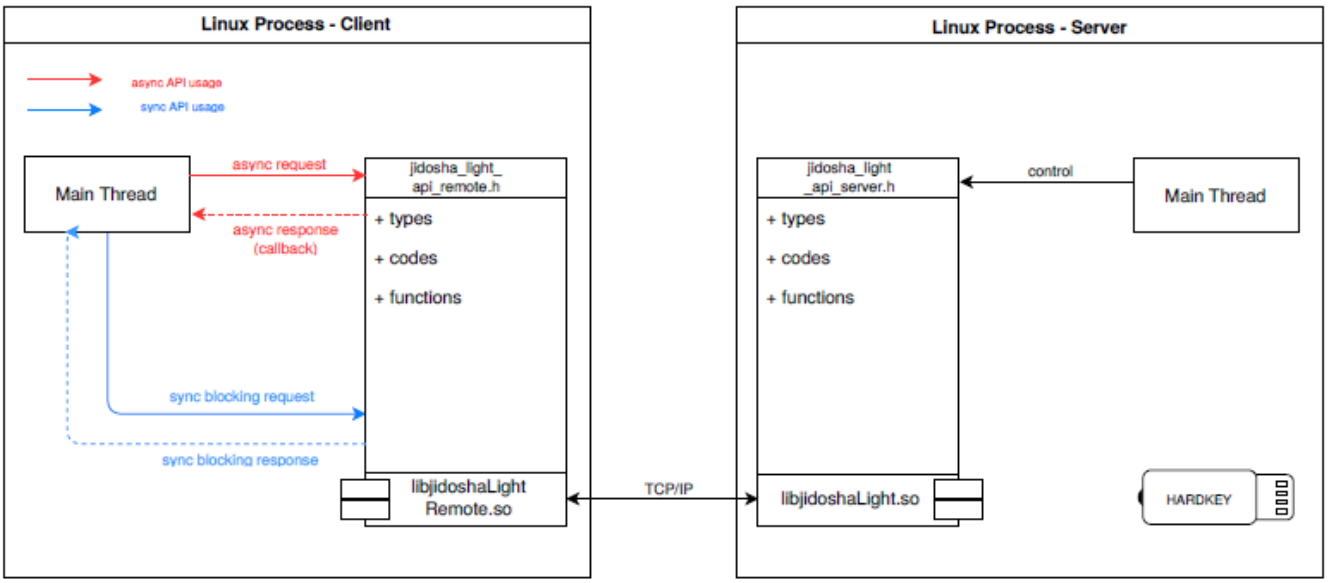


Figura 2 – Diagrama com a arquitetura sugerida para o uso da biblioteca com chamadas remotas

2.1.1 Restrições JidoshaLight Linux

A biblioteca possui suporte a aplicações multithread e multiprocesso, sendo o número máximo de threads de todos os processos limitado pela licença adquirida.

A biblioteca não possui suporte a *fork* de processo.

2.2 Instalação do JidoshaLight Linux

2.2.1 Configuração das permissões do *hardkey*

Para o correto funcionamento do *hardkey* USB, as permissões de acesso do **udev** devem ser alteradas. Para a sua facilidade está incluso no SDK do Jidosha o script `res/scripts/install_udev.sh` executando-o com o argumento `-i` serão instaladas as permissões automaticamente, é possível também rodar o script sem argumento para visualizar as opções. Caso prefira instalar manualmente é necessário seguir os seguintes procedimentos:

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
```

Já para Debian, adicione as linhas:

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

E 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 a Pumatronix Equipamentos Eletrônicos.

2.2.2 Configuração das variáveis de ambiente

Antes de executar as aplicações de teste fornecidas com o SDK, ou qualquer outra aplicação que utilize a biblioteca JidoshaLight Linux, é necessário configurar algumas variáveis de ambiente para o correto funcionamento da biblioteca.

Inicialmente é necessário adicionar o diretório que contém as bibliotecas ao caminho de busca do sistema, como abaixo:

```
$ export LD_LIBRARY_PATH=./lib:$LD_LIBRARY_PATH
```

2.2.2.1 Sistema de log e auditoria

Atenção: a partir da versão 3.3.0 o sistema de log da biblioteca vem DESABILITADO por padrão.

O SDK do JidoshaLight possui um sistema de log que pode ser utilizado para auditar o comportamento da biblioteca em campo. Para habilitar algumas mensagens de depuração pré-configuradas basta exportar a variável de ambiente JL_LOGCFG com o valor "default".

```
$ export JL_LOGCFG=default
```

O sistema de log permite ainda habilitar outras mensagens de depuração e redirecionar o conteúdo destas mensagens para um ou mais arquivos. Esta funcionalidade é configurada através de um arquivo de configuração cuja estrutura é especificada a diante.

A leitura do arquivo de configuração ocorre apenas 1 vez durante a carga da biblioteca e possui a seguinte ordem de busca:
1. caminho absoluto indicado pela variável de ambiente *JL_LOGCFG* (se definida)
2. arquivo *jlog.conf* no diretório atual [./]

Estrutura do arquivo de configuração do sistema de log:

```
# JLog Configuration File
# This is a comment line in a JLog configuration file
# Entry format:
#   TOPIC; LEVEL; TAG_FMT, FILES {comma separated}; SIZES  {comma separated}
#
# Especial Files
#   [STDOUT] - prints to the screen (size always 0)
STDERR    ; CRITICAL      ; SIMPLE_TS    ; [STDOUT], log.txt    ; 0, 20MB
MSGSERVER ; INFO          ; SIMPLE_TS    ; [STDOUT], log.txt    ; 0, 20MB
MSGSERVER ; DEBUG         ; SIMPLE_TS    ; [STDOUT], log.txt    ; 0, 20MB
MSGSERVER ; WARN          ; SIMPLE_TS    ; [STDOUT], log.txt    ; 0, 20MB
MSGSERVER ; NOTICE        ; SIMPLE_TS    ; [STDOUT], log.txt    ; 0, 20MB
MSGSERVER ; CRITICAL      ; SIMPLE_TS    ; [STDOUT], log.txt    ; 0, 20MB
ANPRMSG   ; INFO          ; SIMPLE_TS    ; [STDOUT], log.txt    ; 0, 20MB
ANPRMSG   ; DEBUG         ; SIMPLE_TS    ; [STDOUT], log.txt    ; 0, 20MB
ANPRMSG   ; WARN          ; SIMPLE_TS    ; [STDOUT], log.txt    ; 0, 20MB
ANPRMSG   ; NOTICE        ; SIMPLE_TS    ; [STDOUT], log.txt    ; 0, 20MB
ANPRMSG   ; CRITICAL      ; SIMPLE_TS    ; [STDOUT], log.txt    ; 0, 20MB
LOGGER    ; INFO          ; SIMPLE_TS    ; [STDOUT], log.txt    ; 0, 20MB
LOGGER    ; CRITICAL      ; SIMPLE_TS    ; [STDOUT], log.txt    ; 0, 20MB
LICENSE   ; INFO          ; SIMPLE_TS    ; [STDOUT], log.txt    ; 0, 20MB
LICENSE   ; DEBUG         ; SIMPLE_TS    ; [STDOUT], log.txt    ; 0, 20MB
LICENSE   ; WARN          ; SIMPLE_TS    ; [STDOUT], log.txt    ; 0, 20MB
LICENSE   ; NOTICE        ; SIMPLE_TS    ; [STDOUT], log.txt    ; 0, 20MB
LICENSE   ; CRITICAL      ; SIMPLE_TS    ; [STDOUT], log.txt    ; 0, 20MB
HARDWARE  ; INFO          ; SIMPLE_TS    ; [STDOUT], log.txt    ; 0, 20MB
HARDWARE  ; CRITICAL      ; SIMPLE_TS    ; [STDOUT], log.txt    ; 0, 20MB
JLIB      ; INFO          ; SIMPLE_TS    ; [STDOUT], log.txt    ; 0, 20MB
JLIB      ; CRITICAL      ; SIMPLE_TS    ; [STDOUT], log.txt    ; 0, 20MB
MSGANPR   ; INFO          ; SIMPLE_TS    ; [STDOUT], log.txt    ; 0, 20MB
MSGANPR   ; DEBUG         ; SIMPLE_TS    ; [STDOUT], log.txt    ; 0, 20MB
VLOOP     ; INFO          ; SIMPLE_TS    ; [STDOUT], log.txt    ; 0, 20MB
```

O arquivo de configuração de logs acima fará com que a biblioteca gere todas as mensagens habilitadas, tanto para o arquivo com caminho relativo `*log.txt*` quanto para a saída padrão (stdout). Caso deseje inibir um ou mais tipos de mensagens, basta comentar a linha com '#' ou removê-la.

Para inibir as mensagens ao stdout e escrever apenas no arquivo log.txt, siga o exemplo abaixo para cada tipo de mensagem desejada:

```
MSGANPR   ; INFO          ; SIMPLE_TS    ; log.txt    ; 20MB
```

2.2.3 Configuração do arquivo de preferências

A biblioteca permite uso opcional de um arquivo de preferências. Através desse arquivo é possível configurar os campos do *'struct JidoshaLightConfig'*, sobrescrevendo os campos dessa `*struct*` passados pela API. O formato é json, como segue:

```
{
"jidosha-light" : {
    "config" : {
                "vehicleType" : 3,
                "processingMode" : 4,
                "timeout" : 0,
                "countryCode" : 76,
                "minProbPerChar" : 0.85,
                "maxLowProbabilityChars" : 0,
                "lowProbabilityChar" : "?",
                "avgPlateAngle" : 0.0,
                "avgPlateSlant" : 0.0,
                "maxCharHeight" : 0,
                "minCharHeight" : 0,
                "maxCharWidth" : 0,
                "minCharWidth" : 0,
                "avgCharHeight" : 0,
                "avgCharWidth" : 0,
                "xRoi" : [0,0,0,0],
                "yRoi" : [0,0,0,0],
                "ENABLE_CONFIG_OVERRIDE" : true
            }
    }
}
```

Por padrão, a biblioteca procura o arquivo *jl_anpr_preferences.json* no diretório de trabalho. Caso o arquivo exista, será carregado; caso contrário, será procurado no caminho indicado pela variável de ambiente *JL_ANPR_PREFS*. Caso a variável não exista, não é carregado nenhum arquivo de preferências. Caso exista e o caminho indicado seja um arquivo válido, ele é carregado.

Se um arquivo de preferências foi carregado, as preferências presentes nele só serão aplicadas se o campo `*ENABLE_CONFIG_OVERRIDE*` for `*true*`. Campos da `*struct JidoshaLightConfig*` ausentes do arquivo de preferências receberão o valor default, conforme definido pela biblioteca.

Como o arquivo de preferências é carregado na inicialização da biblioteca (geralmente no início do processo que a utiliza), modificações ao arquivo só terão efeito quando a biblioteca for recarregada. Esse comportamento poderá ser alterado em versões futuras.

2.3 Aplicações exemplo

O SDK inclui algumas aplicações exemplo com código fonte incluso:

- *JidoshaLightSample*: Exemplo de processamento local
- *JidoshaLightSampleClient*: Exemplo de aplicação cliente com processamento remoto assíncrono
- *JidoshaLightSampleServer*: Exemplo de aplicação servidor
- *JidoshaLightSampleAsync*: Exemplo de processamento local assíncrono com múltiplas threads
- *JidoshaLightSampleMulti*: Exemplo de reconhecimento de múltiplas placas na mesma imagem

Caso deseje recompilar os exemplos, utilize o script make_samples.sh:

```
$ cd sample/src && CXX=arm-none-linux-gnueabi-g++ && source make_samples.sh
```

Após configurar o *hardkey* e variáveis de ambiente e conectar o *hardkey* será possível executar os exemplos.

Para executar o *JidoshaLightSample*, execute a partir do terminal o programa de exemplo com a imagem de placa de referência:

```
$ ./sample/bin/JidoshaLightSample ./res/640x480.bmp
```

A aplicação deverá informar a versão da biblioteca bem como o resultado do reconhecimento da imagem.

```
-- JidoshaLight Sample Application --
Library Info
Version: x.y.z
SHA1: abcdefghijklmnopqrstuvwxyz

-> Processing: ./res/640x480.bmp
PLATE: AJK7722 - PROB: 0.9944 - POSITION: (258,338,142,27) - TIME: 419.03 ms
```

Para executar os exemplos *JidoshaLightSampleServer* e *JidoshaLightSampleClient*, execute a partir de um terminal o programa servidor:

```
$ ./sample/bin/JidoshaLightSampleServer
```

A aplicação deverá informar que foi inicializada utilizando a porta TCP 51000 e o *hardkey* foi encontrado.

```
[2016:08:30 15:08:16.620245 : LOGGER   : 0x0001 : INFO] -> Logger session started
Starting server with 1 thread(s), queue size: 10, queueTimeout: 0 ms, 1 connection(s),
port: 51000
[2016:08:30 15:08:16.621808 : MSGSERVER : 0x0001 : INFO] -> Started server at port 51000
[2016:08:30 15:08:16.631327 : HARDWARE : 0x0007 : INFO] -> Hard key attached
[2016:08:30 15:08:17.169088 : HARDWARE : 0x0008 : INFO] -> Found valid hard key
```

Na sequência, em outro terminal, execute o programa cliente. O cliente deverá informar a versão da biblioteca bem como o resultado/estatística do reconhecimento da imagem:

```
$ ./sample/bin/JidoshaLightSampleClient resources/images/640x480.bmp
==================================
Remote API: 127.0.0.1@51000
Threads: 1
Thread queue size: 5
Compilation_Date: Aug 30 2016 - 15:08:10
Images: 1
==================================
PLATE: AJK7722 - PROB:0.9944 - ELAPSED: 14.35 ms - returncode: 0
-- Library --
Version: 2.1.0
Build SHA1: d86e07e560206cb418fdc47b1c5108d7ac76657b
Build FLAGS: I686;Linux_32;DEBUG_LEVEL=DEBUG_LV_LOG;JDONGLE_VENDOR_MODE;...
```

```
-- Total --
TotalTime: 14.35 ms (CPU: 14.35 ms)
Plates: 1
NonEmpty: 1 - 100.00 %
AverageTime: 14.35 ms

-- Load/Decode --
ElapsedTime: 0.83 ms
AverageTime: 0.83 ms (5.77 %)

-- Localization --
ElapsedTime: 7.01 ms
AverageTime: 7.01 ms (48.83 %)

-- Segmentation --
ElapsedTime: 0.69 ms
AverageTime: 0.69 ms (4.81 %)

-- Classification --
ElapsedTime: 5.72 ms
AverageTime: 5.72 ms (39.88 %)
```

Retornando ao terminal do servidor, verifique as mensagens de log adicionais informando dados da licença e eventos de conexão:

```
[2016:08:30 15:11:24.710395 : LICENSE  : 0x0006 : INFO] -> Software license to GAUSSIAN,
max.
threads 16, max. connections 16
[2016:08:30 15:11:24.710421 : LICENSE  : 0x0001 : INFO] -> Valid license found 0x2137069056
[2016:08:30 15:11:24.857709 : MSGSERVER : 0x0005 : INFO] -> Accepted connection:
127.0.0.1@51000
[2016:08:30 15:11:25.563783 : MSGSERVER : 0x0007 : NOTICE] -> Dropped connection:
127.0.0.1:@51000
```

3 JidoshaLight Windows

A biblioteca de software *JidoshaLight Linux* foi criada para funcionar em conjunto com o *hardkey* (chave de segurança) que acompanha a biblioteca. Ou seja, para o correto funcionamento da biblioteca o referido *hardkey* deverá estar conectado à USB do ambiente em que a biblioteca será utilizada. Existem duas versões de *hardkey*, uma para uso geral e a versão de demonstração, com data de validade. Quando a data de validade expira, a biblioteca automaticamente passa a retornar placas vazias. Se o *hardkey* de demonstração expirar, é possível comprar uma licença ou estender o período de demonstração, entrando em contato com a Pumatronix.

Verifique os pré-requisitos de instalação explicitados no Manual de Produto.

Instalação do JidoshaLight Windows

Para a instalação somente há a necessidade de plugar o *hardkey* em uma máquina windows que executará o software, em seguida o windows deverá instalar um driver automaticamente na primeira vez. Para se testar a instalação ocorreu corretamente pode-se executar as aplicações exemplo, detalhadas em Configuração do arquivo de preferências.

3.1.1 Configuração das variáveis de ambiente

Antes de executar as aplicações de teste fornecidas com o SDK, ou qualquer outra aplicação que utilize a biblioteca JidoshaLight Windows, é necessário configurar algumas variáveis de ambiente para o correto funcionamento da biblioteca.

Inicialmente é necessário adicionar o diretório que contém as bibliotecas ao caminho de busca do sistema, para isso acesse a pasta do SDK e digite o comando:

```
$ set PATH=./lib;%PATH%
```

NOTA: O comando anterior somente alterará o PATH para a sessão de terminal aberta, caso precise configurar para o ambiente é necessário acessar o painel de controle > sistema > alterar configurações > propriedades do sistema > avançado > variáveis de ambiente e lá alterar o valor da variável de sistema, ou de usuário, chamada Path.

3.1.1.1 Sistema de log e auditoria

Ver Sistema de log e auditoria utilizada no JidoshaLight Linux.

3.1.2 Configuração do arquivo de preferências

Ver Configuração do arquivo de preferências utilizada no JidoshaLight Linux.

3.2 Aplicações exemplo

O SDK inclui algumas aplicações exemplo com código fonte incluso:

- *JidoshaLightSample*: Exemplo de processamento local
- *JidoshaLightSampleClient*: Exemplo de aplicação cliente com processamento remoto assíncrono
- *JidoshaLightSampleServer*: Exemplo de aplicação servidor
- *JidoshaLightSampleAsync*: Exemplo de processamento local assíncrono com múltiplas threads
- *JidoshaLightSampleMulti*: Exemplo de reconhecimento de múltiplas placas na mesma imagem
- *JidoshaLightSampleServerService*: Exemplo de servidor como serviço do Windows

Para rodar o *JidoshaLightSample*, da pasta do SDK execute o comando abaixo e deve obter uma saída parecida:

```
>sample\bin\JidoshaLightSample.exe .\res\640x480.bmp
[ano:mes:dia horario : LOGGER   : 0x0001 : INFO] -> JLib log session started
[ano:mes:dia horario : JLIB     : 0x0004 : INFO] -> JLib singleton created

-- JidoshaLight LPR Sample Application - 64 bits --
Compilation Date: mes dia ano horario
Library Info
Version: x.y.z
SHA1: sha1
[ano:mes:dia horario : HARDWARE : 0x0008 : INFO] -> Hardkey attached
[ano:mes:dia horario : HARDWARE : 0x000A : INFO] -> Hardkey access valid
[ano:mes:dia horario : LICENSE  : 0x0002 : INFO] -> Licensed to empresa, product LPR,
threads 4, connections 1, serial 150089957 (0x8f230e5), TTL: -1
-- LicenseInfo --
>> Serial: 0x8f230e5
>> Customer: empresa
```

```
>> State: 0
>> TTL: -1 hours
>> MaxThreads: 4
>> MaxConections: 1
FILE: ..\..\res\640x480.bmp - PLATE: AJK7722 - COUNTRY: 76 - PROB: 0.9912 - POSITION:
(258,339,142,25) - TIME: 12.41 ms

Exiting
[ano:mes:dia horario : JLIB     : 0x0002 : INFO] -> JLib network module stopped
[ano:mes:dia horario : JLIB     : 0x0005 : INFO] -> JLib singleton destroyed
[ano:mes:dia horario : LOGGER   : 0x0002 : INFO] -> JLib log session stopped
```

4 JidoshaLight Linux/FPGA

A biblioteca de software *JidoshaLight Linux* com aceleração por FPGA é licenciada a partir de um arquivo de licença atrelado ao hardware, sem a necessidade de uso de *hardkey* (chave de segurança). Esta biblioteca possui suporte para aceleração em hardware baseado em FPGAs Xilinx da família **Zynq-7000**. Por padrão possui suporte para o dispositivo *XC7Z020-CLG400*, podendo ser adaptada para dispositivos de maior capacidade.

Verifique os pré-requisitos de instalação explicitados no Manual de Produto.

4.1 Arquitetura de software do JidoshaLight Linux/FPGA

A arquitetura de software é semelhante à da versão Linux sem aceleração, descrita em JidoshaLight Linux.

As principais diferenças estão nas interfaces adicionais de programação do dispositivo e na área reservada de memória compartilhada (*shared memory*). O detalhamento de configuração e instalação são específicas e estão descritos em Instalação.

As figuras a seguir ilustram a arquitetura sugerida tanto para API local como remota.

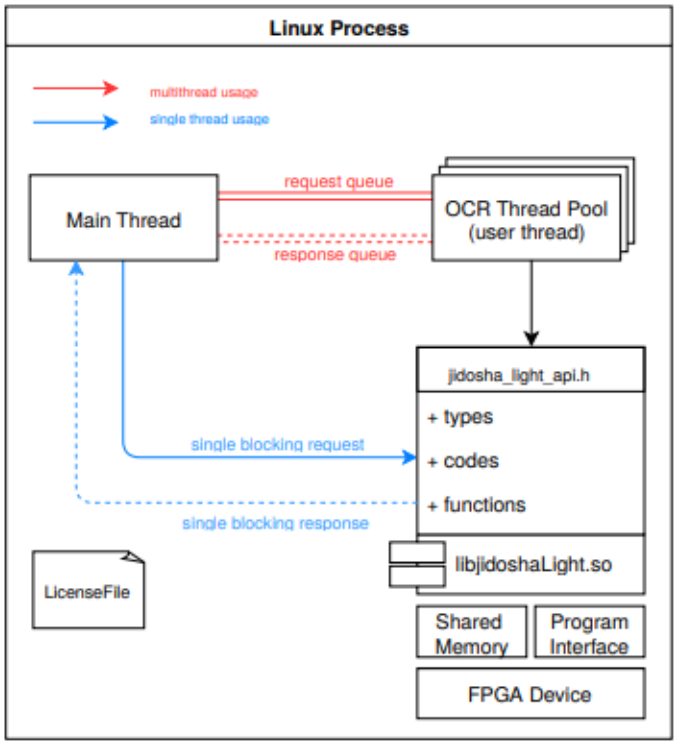


Figura 3 – Diagrama com casos de uso da API local

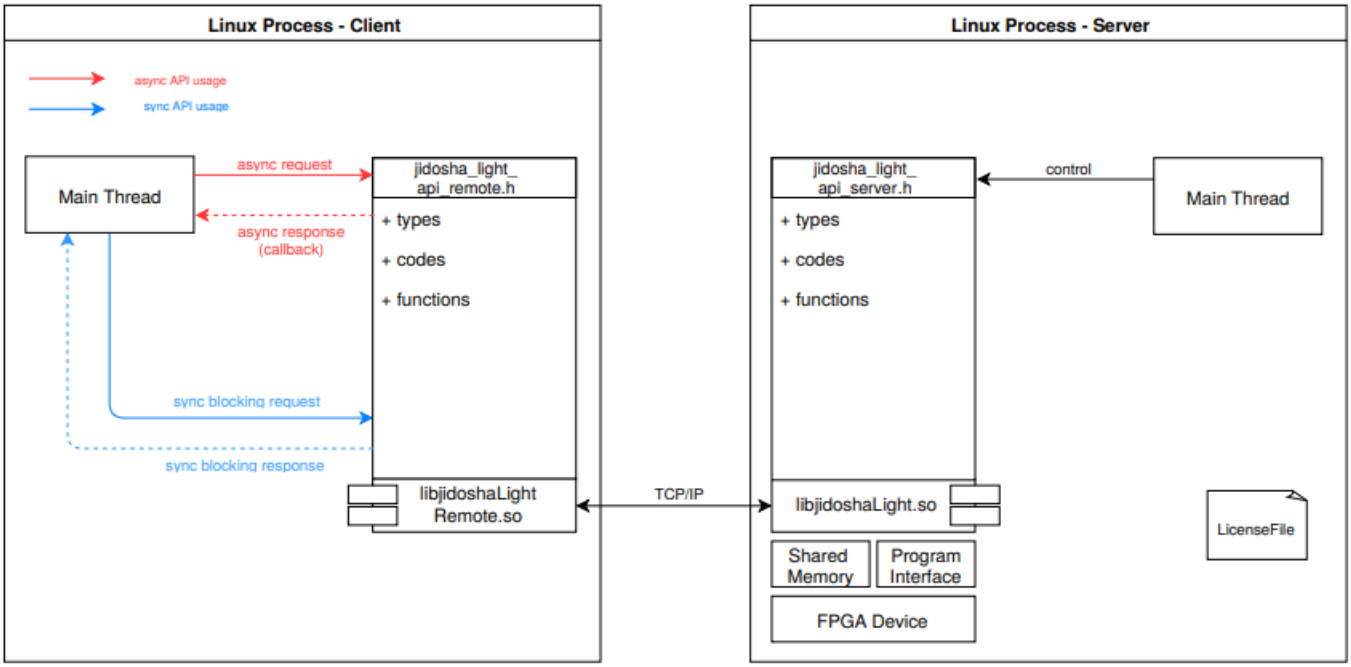


Figura 4 – Diagrama com os casos de uso da API remota

4.1.1 Restrições JidoshaLight Linux/FPGA

A biblioteca com aceleração por FPGA possui suporte a aplicações multithread, sendo o número máximo de threads limitado pela licença adquirida. Não existe suporte para aplicações multiprocesso.

4.2 Instalação do JidoshaLight Linux/FPGA

4.2.1 Configuração da licença

A biblioteca é licenciada através de um arquivo atrelado ao dispositivo utilizado.

Para obter o identificador do seu hardware é necessário executar a aplicação *JidoshaLightDna* a partir do terminal do dispositivo devidamente configurado como descrito em Configuração das variáveis de ambiente.

```
$ ./tools/JidoshaLightDna
0xFEDCBA9876543210
```

IMPORTANTE: O uso concorrente desta aplicação com qualquer outra que utilize a biblioteca não é permitido e pode causar travamentos.

4.2.2 Configuração das variáveis de ambiente

Antes de executar as aplicações de teste fornecidas com o SDK, ou qualquer outra aplicação que utilize a biblioteca JidoshaLight Linux, é necessário configurar algumas variáveis de ambiente para o correto funcionamento da biblioteca.

Ao utilizar a versão com aceleração por FPGA, além das variáveis descritas em Configuração das variáveis de ambiente, as configurações descritas a seguir são necessárias:

- JL_MPOOL_BASE: Endereço base de memória destinado à comunicação entre biblioteca e FPGA. Caso não seja definido, o valor padrão é 0x3A000000. Ex.:

```
$ export JL_MPOOL_BASE=0x3A000000
```

- JL_MPOOL_BUFFERNUM: Número de *buffers* de memória necessários para execução da biblioteca. Caso não seja definido, o valor padrão é de 32 *buffers*, sendo o valor mínimo necessário 12 *buffers* por thread que utiliza a biblioteca de forma concorrente. Ex.:

```
$ export JL_MPOOL_BUFFERNUM=32
```

- JL_MPOOL_BUFFERSIZE: Tamanho de cada **buffer** de memória, sendo necessário ser múltiplo de 4096 **bytes*. Caso não seja definido, o valor padrão é 2097152 **bytes* (2MB). Este é o valor necessário para processar imagens de até 800x600 pixels. Este valor precisa ser superior à 4 vezes a resolução da imagem. Ex.:

```
$ export JL_MPOOL_BUFFERSIZE=2097152
```

- JL_LICENSE_FILE: Caminho para o arquivo de licença. O arquivo de licença é atrelado ao dispositivo utilizado. Para obter o identificador, consulte Configuração da licença. Ex.:

```
$ export JL_LICENSE_FILE=./license.bin
```

4.2.3 Configuração do kernel Linux

Para comunicação entre a biblioteca e o dispositivo FPGA são necessárias duas interfaces, uma dedicada para configuração da FPGA e uma memória compartilhada para troca de dados.

A interface de configuração é fornecida pela Xilinx através de um *char device* (/dev/xdevcfg) e não é atualmente parte do *kernel* Linux padrão. Código fonte e instruções para instalação podem ser consultados na página Wiki da Xilinx.

A memória compartilhada precisa ser visível em */dev/mem* e reservada para uso exclusivo pela biblioteca, não podendo ser utilizada pelo **kernel** Linux.

Para tanto, é necessário limitar a quantidade de memória utilizada pelo **kernel** ao inicializá-lo.

Abaixo segue um exemplo para reservar os últimos 96MB de memória de um dispositivo com 1GB de memória RAM. No u-boot, configurar:

```
set bootargs 'root=/dev/ram mem=928M rw'
```

Para disponibilizar o dispositivo */dev/mem*, utilize a opção *CONFIG_DEVMEM=y* no kconfig no processo de compilação do *kernel*.

Adicione também ao Linux *device tree* (DTS) as seguintes configurações:

```
memory {
  device_type = "memory";
  reg = <0x3A000000 0x6000000>;
};

reserved-memory {
  #address-cells = <1>;
  #size-cells = <1>;
  ranges;

  linux,cma {
    compatible = "shared-dma-pool";
    reusable;
    size = 0x6000000;
    alignment = 0x1000;
    linux,cma-default;
  };
};
```

4.3 Aplicações exemplo

Ver Aplicações exemplo utilizadas para o JidoshaLight Linux.

5 JidoshaLight Android™

A biblioteca de software *JidoshaLight Android™* foi criada para funcionar em conjunto com o arquivo de licença que deve ser gerado após a instalação da aplicação pelo usuário. O arquivo de licença é gerado por instalação e é atrelado ao hardware do dispositivo, sendo necessário um novo licenciamento no caso de reinstalação da aplicação ou modificação do hardware do aparelho, incluindo o SIM card do dispositivo. A substituição da bateria não acarreta na necessidade de uma nova licença. Para licenças temporárias, liberadas por tempo limitado, a data e hora do equipamento devem estar sincronizadas com a rede de celular.

A biblioteca possui suporte a aplicações multithread, sendo o número **máximo de threads** e o **mínimo tempo de processamento** limitados pela licença adquirida. Para o uso da API servidor, o número **máximo de conexões simultâneas** que este aceita também é limitado pela licença.

As funcionalidades da biblioteca *JidoshaLight Android* são acessadas através da API Java. A presente versão possui compatibilidade com processadores ARM™ (armv7-a) com sistema operacional Android™ 4.4 ou superior para o uso da biblioteca (shared libraries e classes Java básicas) e Android™ 8 ou superior para a instalação do aplicativo de demonstração.

5.1 Arquitetura de software do JidoshaLight Android™

A forma mais recomendada de trabalhar com a biblioteca JidoshaLight em plataforma Android é através da topologia *cliente assíncrono e servidor*. Esta topologia permite otimizar o fluxo do processo de reconhecimento de placas, uma vez que todo o processamento e alocação de memória acontece em código nativo. Esta topologia ainda possibilita processar as imagens externamente sem alterações na aplicação. A aplicação de demonstração que acompanha o SDK implementa esta topologia.

Usualmente uma melhor experiência de uso é obtida em modo *freeflow*. No modo *freeflow*, em oposição ao *point and shoot* (apontar e disparar), o processo de reconhecimento de placa é feito em todas as imagens enviadas pela câmera, sem necessidade de intervenção (disparo) por parte do usuário. Assim, logo que a câmera for acionada o processamento começa e callbacks com os resultados de reconhecimento vão sendo geradas de forma assíncrona. A topologia **cliente assíncrono e servidor** está apta a trabalhar em modo *freeflow* sem nenhuma mudança significativa na implementação da aplicação.

Pontos **positivos** do cliente assíncrono freeflow:

1. Simplificação do código da aplicação, facilitando a integração da biblioteca
2. Melhor experiência de uso (reconhecimentos mais rápidos)
3. Gestão automática de recursos (filas, threads, rede)
4. Maior desacoplamento entre a aquisição de imagem (câmera), o processamento (LPR) e a saída (UI e DB)
5. Capacidade de processamento local ou remoto sem modificação do código fonte

Pontos **negativos** do cliente assíncrono freeflow:

1. Callbacks ocorrem em thread separada à thread da UI, sendo necessária sincronização (runOnUiThread)
2. Callbacks são emitidas sequencialmente e não podem bloquear (código da callback deve ser leve e rápido)
3. Tratamento de erros assíncrono é normalmente mais complexo

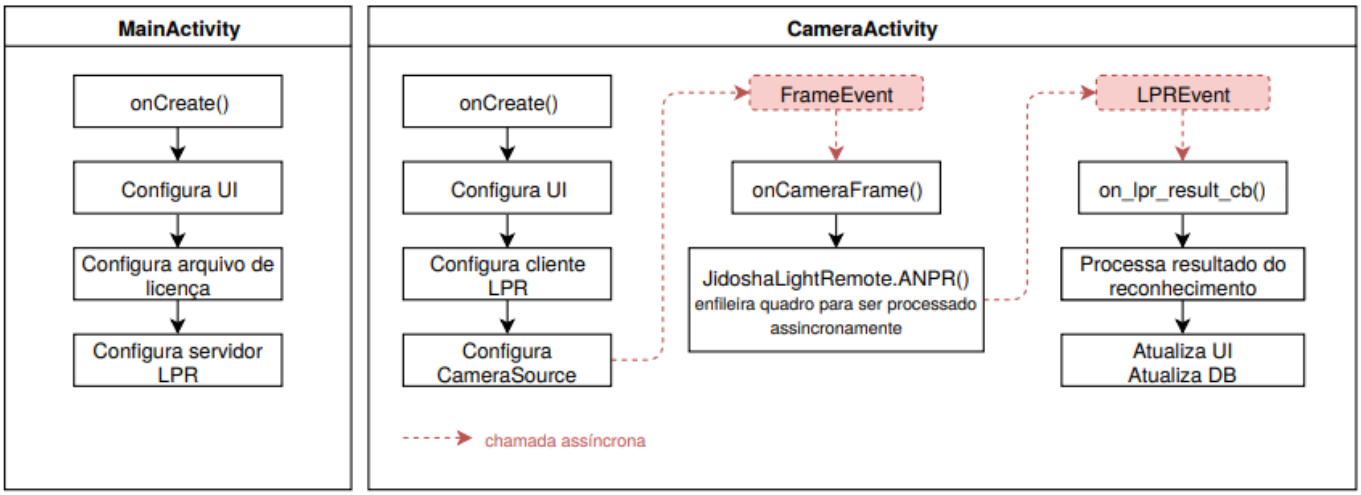


Figura 5 - Exemplo de fluxo para uma aplicação cliente assíncrono freeflow: MainActivity configura a licença da biblioteca e inicia o servidor de processamento, CameraActivity configura o cliente de leitura de placas, a camera (CameraSource) e trata os eventos

5.1.1 Restrições JidoshaLight Android™

ATENÇÃO: As restrições de memória apresentadas à seguir não se aplicam à memória alocada nativamente (dentro da shared library).
Para aplicações de alto desempenho, recomenda-se o uso da API cliente assíncrono.

O sistema operacional Android™ possui restrições rigorosas quanto ao uso de memória RAM pelas aplicações. O valor máximo de memória que uma aplicação pode alocar na *heap* varia entre dispositivos, mas fica em torno de 24MB a 36MB. Se uma aplicação tentar alocar mais memória do que lhe é permitido ocorre uma exceção de `*OutOfMemoryError*` e a aplicação é finalizada pelo sistema operacional.

Uma vez que a resolução das câmeras dos smartphones e tablets está cada vez maior, o desenvolvedor deve atentar ao tamanho das imagens que ele pretende reconhecer a fim de mitigar a possibilidade de uma exceção do tipo `*OutOfMemoryError*`. Por exemplo, uma imagem de 8MP em formato Bitmap ARGB8888 ocupa 24MB, o que já seria suficiente para superar o limite de memória em vários dispositivos.

Como o JidoshaLight precisa que **os caracteres da placa tenham no máximo 30 pixels de altura**, uma imagem com resolução de **1280x720** é suficiente para fins de reconhecimento de placa. Caso se queira exibir uma imagem de alta resolução para o usuário, pode-se adquirir e armazenar a imagem em alta resolução e para processamento utilizar-se dos métodos de decodificação com redução de resolução suportados pela classe `*android.graphics.Bitmap*` do Android™. Neste caso, deve-se levar em conta a redução dos caracteres no processo de redução do tamanho da imagem, garantindo o tamanho de 15 a 30 pixels na imagem reduzida.

Outra especificidade importante do sistema operacional Android™ é relacionada ao travamento da thread da interface gráfica. Por padrão, a thread da interface gráfica é a única criada pelo aplicativo e todo o processamento é realizado nela. Para que a interface gráfica continue responsiva às ações do usuário, ela não deve bloquear por mais de alguns poucos milissegundos. Se isso ocorrer, o sistema operacional irá emitir um alerta ao usuário relatando que a aplicação parou de responder ou simplesmente irá interromper o aplicativo. Para maiores informações sobre o gerenciamento de memória no Android:

- https://developer.android.com/training/articles/memory.html
- https://developer.android.com/training/displaying-bitmaps/index.html?hl=pt-br

5.2 Instalação do JidoshaLight Android™

ATENÇÃO: Todas as classes Java que possuem métodos marcados como `native`não podem ter seu package alterado.
Todas as demais classes podem ser movidas livremente.

O SDK de desenvolvimento da biblioteca de leitura das placas veiculares *JidoshaLight* para Android acompanha a API e as shared libraries nativas linguagem C (compartilhadas, que podem ser acessadas por qualquer código JNI), os wrappers e bibliotecas para interface Java e uma aplicação de demonstração descrita em Aplicativo Exemplo.

5.2.1 Licenciamento

Um arquivo de licença válido é necessário para o funcionamento da biblioteca *JidoshaLight* em sistemas Android™. O licenciamento é feito por dispositivo e por tempo limitado, sendo necessário um novo licenciamento caso o equipamento tenha suas características de hardware alteradas ou o tempo de vigência da licença tenha acabado.

A API `JidoshaLight.setLicenseFromData()` deve ser utilizada para passar o conteúdo do arquivo de licença para a biblioteca e deve ser feito **antes** de qualquer outra chamada a outras funções da API. A classe `JidoshaLightAndroidHelper.java` traz ainda algumas funções utilitárias que auxiliam no processo de carga da licença.

O procedimento de requisição de licença é totalmente automatizado pela função `JidoshaLight.getLicenseFromServer`, sendo necessário apenas que o usuário cadastre o *Device ID* do dispositivo junto à Pumatronix. A classe `LicenseManagerFragment.java` da aplicação de exemplo mostra como requisitar o *Device ID* do equipamento e como solicitar uma licença do servidor.

Para maiores informações sobre licenciamento de dispositivos entre em contato com o Suporte Técnico.

5.2.2 Permissões

As seguintes permissões são necessárias para o funcionamento da biblioteca e devem ser incluídas no `*AndroidManifest.xml*` da aplicação:

```
<!-- Permissões necessárias para usar a biblioteca (obrigatório) -->
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
<!-- Permissões necessárias para usar a câmera do dispositivo (opcional) -->
<uses-feature android:name="android.hardware.camera" android:required="true" />
<uses-permission android:name="android.permission.CAMERA" />
<!-- Permissões necessárias para usar a câmera MJPEG (opcional) -->
<uses-permission android:name="android.permission.INTERNET"/>
```

5.3 Aplicativo Exemplo

A aplicação de exemplo que acompanha o SDK foi desenvolvida para uso com Android™ Studio 4 ou superior. Ela mostra como utilizar a biblioteca para reconhecer as placas de um fluxo de vídeo proveniente da câmera traseira do celular ou de uma câmera MJPEG externa. Ela traz também um exemplo de implementação para a tela de configuração dos parâmetros da biblioteca, Activity de câmera com suporte

a grade e zoom, lista de reconhecimentos, detalhes do reconhecimento, sistema de licenciamento e de integração com banco de dados para busca por informações relacionadas à placa detectada.

5.3.1 Package `br.gaussian.io`

- *Mjpeg.java*: Classe wrapper sobre a API nativa do decoder MJPEG. Veja a classe `MjpegCamera.java` para uma implementação de mais alto nível.

5.3.2 Package `br.gaussian.jidoshalight`

- *JidoshaLight.java*: Classe contendo as funções da API local (reconhecimento de placas e licenciamento) e códigos de retorno de função
- *JidoshaLightRemote.java*: Classe contendo as funções da API remota assíncrona
- *JidoshaLightServer.java*: Classe contendo as funções da API servidor

5.3.3 Package `br.gaussian.jidoshalight.camera`

- *BaseCameraSource.java*: Classe base para todas as implementações de câmera
- *BackCamera.java*: Implementação para a câmera traseira do smartphone
- *MjpegCamera.java*: Implementação para uma câmera MJPEG externa
- *CameraFrame.java*: Classe que armazena um quadro de uma câmera
- *CameraView.java*: View capaz de exibir um fluxo de imagens de uma BaseCameraSource; provê suporte a grade, overlay para placas, zoom por pinça e seleção de ROI

5.3.4 Package `br.gaussian.jidoshalight.sample`

5.3.4.1 Common

- *common/JidoshaLightAndroidHelper.java*: Classe auxiliar contendo métodos de suporte para o processo de leitura e escrita do arquivo de licença, além de outros métodos utilitários.
- *common/JidoshaLightServerHelper.java*: Classe auxiliar contendo métodos de suporte para a incialização do servidor LPR local.

5.3.4.2 Activities

- *MainActivity*: Activity principal da aplicação, mostra como configurar o arquivo de licença e inicializar o servidor local de leitura de placas.

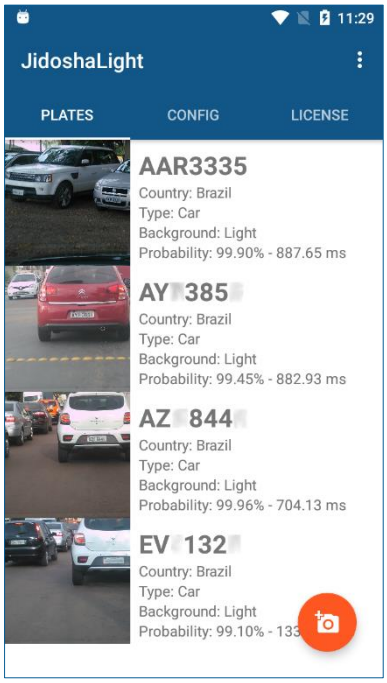


Figura 6 – MainActivity com caracteres ofuscados por questão de privacidade

- *DetailActivity*: Activity acionada ao selecionar um item da lista de reconhecimento. Expande as informações de um determinado reconhecimento.

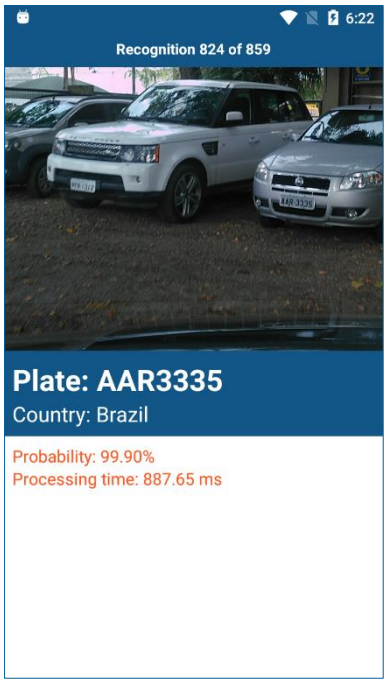


Figura 7 - DetailActivity

- *CameraActivity*: Exemplifica o processo de configuração e captura de imagens da câmera, permitindo:

6) Instanciar uma câmera a partir das configurações;
7) Configurar um cliente de processamento de placas;
8) Configurar o zoom óptico através do movimento de pinça;
9) Selecionar a região de interesse (ROI) através do toque;
10) Habilitar/desabilitar o processamento.

Para um melhor desempenho de reconhecimento, o foco e o zoom da câmera devem permitir capturar imagens com boa nitidez e tamanho. A altura da placa deve ficar entre 30 e 50 pixels. As guias têm o objetivo de auxiliar no enquadramento da placa, garantindo o tamanho e a orientação correta da placa no momento da captura. A placa deve ter aproximadamente o tamanho de um retângulo da grade.

Pelo fato de a câmera do *smartphone* ou *tablet* estar em constante movimento, é ideal, quando existente, ativar os recursos de auto-foco e de estabilização de vídeo do aparelho. Dependendo da aplicação, é recomendável exportar ajustes de foco e exposição manuais para um melhor resultado.

Figura 8 – CameraActivity: A altura ideal da placa para o reconhecimento deve ser a mesma de um retângulo da grade (a placa não precisa estar alinhada com a grade)

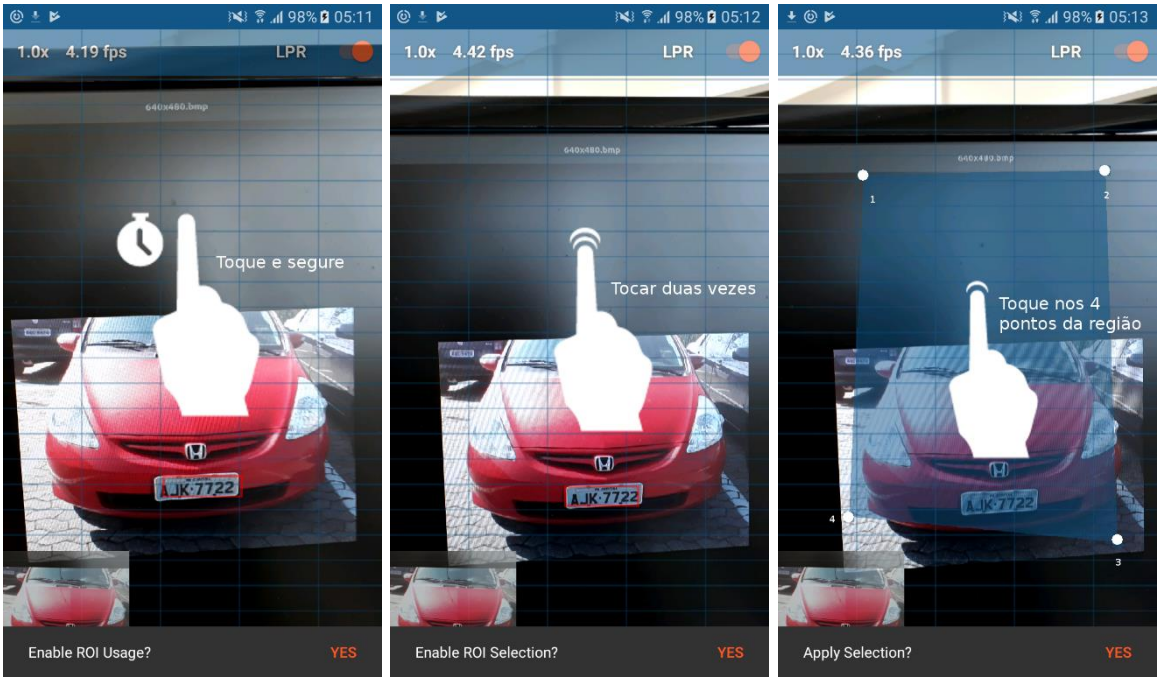
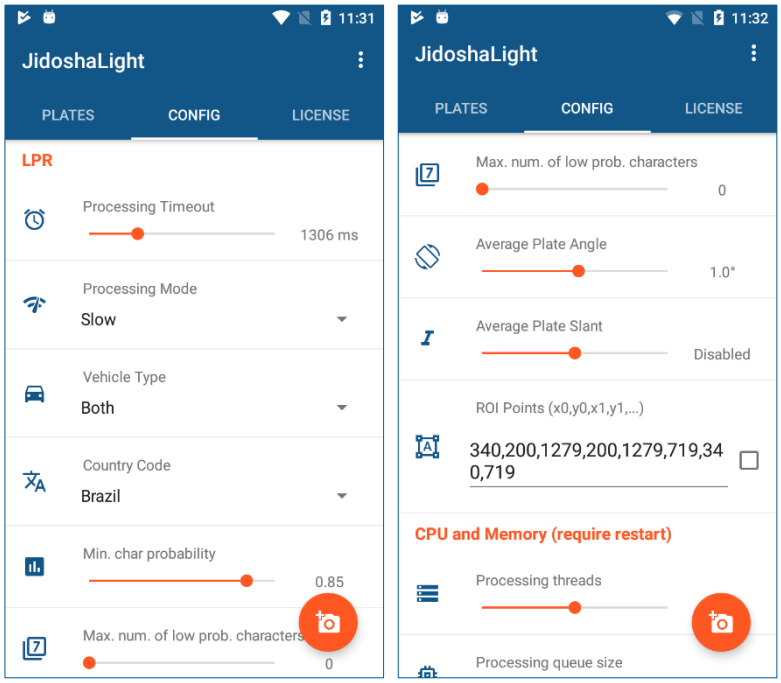


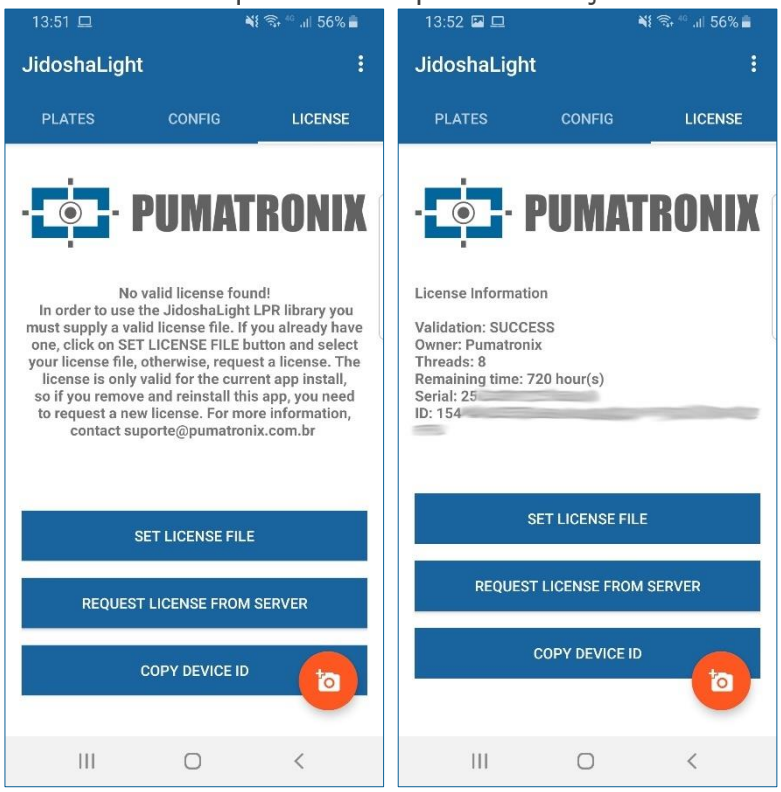
Figura 9 - Seleção da região da ROI: 1) Toque e segure na tela para habilitar o uso da ROI, 2) Toque duas vezes para iniciar a seleção dos pontos da ROI, 3) Toque nos quatros pontos que delimitam a região (em sentido horário)

5.3.4.3 Fragments

- *ConfigurationFragment*: Fragment utilizado para exibir e configurar os parâmetros da biblioteca JidoshaLight. Os parâmetros configuráveis são os mesmos disponíveis na API Java/C



- *LicenseManagerFragment*: Fragment utilizado para implementar o sistema de licenciamento. Mostra como ler o **Device ID** e como requisitar um arquivo de licença do servidor



A mudança de algumas configurações e do arquivo de licença requerem que a aplicação seja reiniciada.

6 APIs de usuário

A biblioteca *JidoshaLight* exporta 4 APIs distintas para o reconhecimento automático de placas:

- *Local*
- *Remota Síncrona*
- *Remota Assíncrona*
- *Servidor.*

A API Remota Síncrona será descontinuada no futuro e não deve ser utilizada em novos projetos. Além das APIs de reconhecimento, a biblioteca provê algumas funções utilitárias, como um receptor de vídeo no formato MJPEG e um leitor de licenças. Estas APIs suplementares estão parcialmente documentadas neste manual. Para maiores informações quanto ao uso, consulte os headers na pasta `include/gaussian/common`.

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 dúvida ou suporte a outras linguagens, entre em contato com o Suporte Técnico da Pumatronix.

6.1 Limitações conhecidas

Seguem as limitações conhecidas da biblioteca *JidoshaLight*:

11) Quando a biblioteca é carregada dinamicamente (*LoadLibrary* no Windows, *dlopen* no Linux), a mesma não poderá ser descarregada (*FreeLibrary* e *dlclose*, respectivamente).
12) No Linux, o processo onde roda a biblioteca não pode sofrer *fork* (cópia de processo).
13) No caso de uso concorrente e no mesmo processo do *JidoshaLight* com a biblioteca *Jidosha Portuário* (container) ou *Jidosha Ferroviário* (rail), o JidoshaLight deve ser carregado por último. Essa limitação será removida em uma versão futura das bibliotecas.

6.2 API JidoshaLight 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.

6.2.1 API JidoshaLight C/C++ (Local)

A API Local contém os tipos, as definições e as funções básicas para o processamento local das imagens. Desde o release 2.4.4 seu conteúdo está dividido entre os arquivos *jidosha_light_api_common.h* e *jidosha_light_api.h*.

Para manter a compatibilidade com as versões anteriores, o header *jidosha_light_api_common.h* `é incluído pelo `*jidosha_light_api.h*`.

6.2.1.1 jidosha_light_api_common.h

```
//==============================================================================
//  CODES
//==============================================================================
```

```
enum JidoshaLightVehicleType {
    JIDOSHA_LIGHT_VEHICLE_TYPE_CAR                      = 1,
    JIDOSHA_LIGHT_VEHICLE_TYPE_MOTO                     = 2,
    JIDOSHA_LIGHT_VEHICLE_TYPE_BOTH                     = 3
};

enum JidoshaLightMode {
    JIDOSHA_LIGHT_MODE_DISABLE                          = 0,
    JIDOSHA_LIGHT_MODE_FAST                             = 1,
    JIDOSHA_LIGHT_MODE_NORMAL                           = 2,
    JIDOSHA_LIGHT_MODE_SLOW                             = 3,
    JIDOSHA_LIGHT_MODE_ULTRA_SLOW                       = 4,

    /* the following values can be added to one of the above modes */
    JIDOSHA_LIGHT_LOCALIZATION_MODE_0                   = 0 << 8,
    JIDOSHA_LIGHT_LOCALIZATION_MODE_1                   = 1 << 8,
    JIDOSHA_LIGHT_LOCALIZATION_MODE_2                   = 2 << 8
};

/* ISO 3166-1 */
enum JidoshaLightCountryCode {
    JIDOSHA_LIGHT_COUNTRY_CODE_CONESUL                  = 0,
    JIDOSHA_LIGHT_COUNTRY_CODE_ARGENTINA                = 32,
    JIDOSHA_LIGHT_COUNTRY_CODE_BRAZIL                   = 76,
    JIDOSHA_LIGHT_COUNTRY_CODE_CHILE                    = 152,
    JIDOSHA_LIGHT_COUNTRY_CODE_COLOMBIA                 = 170,
    JIDOSHA_LIGHT_COUNTRY_CODE_MEXICO                   = 484,
    JIDOSHA_LIGHT_COUNTRY_CODE_PARAGUAY                 = 600,
    JIDOSHA_LIGHT_COUNTRY_CODE_PERU                     = 604,
    JIDOSHA_LIGHT_COUNTRY_CODE_URUGUAY                  = 858,
    JIDOSHA_LIGHT_COUNTRY_CODE_NETHERLANDS              = 528,
    JIDOSHA_LIGHT_COUNTRY_CODE_FRANCE                   = 250
};

enum JidoshaLightReturnCode {
    /* success */
    JIDOSHA_LIGHT_SUCCESS                               = 0,
    /* basic errors */
    JIDOSHA_LIGHT_ERROR_FILE_NOT_FOUND                  = 1,
    JIDOSHA_LIGHT_ERROR_INVALID_IMAGE                   = 2,
    JIDOSHA_LIGHT_ERROR_INVALID_IMAGE_TYPE              = 3,
    JIDOSHA_LIGHT_ERROR_INVALID_PROPERTY                = 4,
    JIDOSHA_LIGHT_ERROR_COUNTRY_NOT_SUPPORTED           = 5,
    JIDOSHA_LIGHT_ERROR_API_CALL_NOT_SUPPORTED          = 6,
    JIDOSHA_LIGHT_ERROR_INVALID_ROI                     = 7,
    JIDOSHA_LIGHT_ERROR_INVALID_HANDLE                  = 8,
    JIDOSHA_LIGHT_ERROR_API_CALL_HAS_NO_EFFECT          = 9,
    JIDOSHA_LIGHT_ERROR_INVALID_IMAGE_SIZE              = 10,
    /* license errors */
    JIDOSHA_LIGHT_ERROR_LICENSE_INVALID                 = 16,
    JIDOSHA_LIGHT_ERROR_LICENSE_EXPIRED                 = 17,
    JIDOSHA_LIGHT_ERROR_LICENSE_MAX_THREADS_EXCEEDED    = 18,
    JIDOSHA_LIGHT_ERROR_LICENSE_UNTRUSTED_RTC           = 19,
    JIDOSHA_LIGHT_ERROR_LICENSE_MAX_CONNS_EXCEEDED      = 20,
    JIDOSHA_LIGHT_ERROR_LICENSE_UNAUTHORIZED_PRODUCT    = 21,
    /* others */
    JIDOSHA_LIGHT_ERROR_OTHER                           = 999
```

```
};

enum JidoshaLightReturnCodeNetwork {
    /* network errors */
    JIDOSHA_LIGHT_ERROR_SERVER_CONNECT_FAILED          = 100,
    JIDOSHA_LIGHT_ERROR_SERVER_DISCONNECTED            = 101,
    JIDOSHA_LIGHT_ERROR_SERVER_QUEUE_TIMEOUT           = 102,
    JIDOSHA_LIGHT_ERROR_SERVER_QUEUE_FULL              = 103,
    JIDOSHA_LIGHT_ERROR_SOCKET_IO_ERROR                = 104,
    JIDOSHA_LIGHT_ERROR_SOCKET_WRITE_FAILED            = 105,
    JIDOSHA_LIGHT_ERROR_SOCKET_READ_TIMEOUT            = 106,
    JIDOSHA_LIGHT_ERROR_SOCKET_INVALID_RESPONSE        = 107,
    JIDOSHA_LIGHT_ERROR_HANDLE_QUEUE_FULL              = 108,
    JIDOSHA_LIGHT_ERROR_SERVER_CONN_LIMIT_REACHED      = 213,
    JIDOSHA_LIGHT_ERROR_SERVER_VERSION_NOT_SUPPORTED   = 214,
    JIDOSHA_LIGHT_ERROR_SERVER_NOT_READY               = 215
};

/* Raw image pixel format */
enum JidoshaLightRawImgFmt {
    JIDOSHA_LIGHT_IMG_FMT_XRGB_8888                    = 0,
    JIDOSHA_LIGHT_IMG_FMT_RGB_888                      = 1,
    JIDOSHA_LIGHT_IMG_FMT_LUMA                         = 2,
    JIDOSHA_LIGHT_IMG_FMT_YUV420                       = 3
};

//==============================================================================
//  TYPES
//==============================================================================
//  JidoshaLightConfig
//==============================================================================
typedef struct JidoshaLightConfig
{
    int configId;                       // Unique Configuration ID
    int vehicleType;                    // Vehicle type
    int processingMode;                 // Processing Mode
    int timeout;                        // Processing timeout in milliseconds
    int countryCode;                    // Plate Syntax Country

    float minProbPerChar;               // Range [0,1] - Minimal probability to accept a
                                        // given character recognition
    int   maxLowProbabilityChars;       // Max number of characters whose propability is
lower
                                        // than minProbPerChar to accept a recognition
    char  lowProbabilityChar;           // ASCII encodded character that will replace
characters
                                        // with probability lower than minProbPerChar
    float avgPlateAngle;                // Average plate angle
    float avgPlateSlant;                // Average plate slant
    int   maxCharHeight;                // Max acceptable char height in pixels (0 ==
default value)
    int   minCharHeight;                // Min acceptable char height in pixels (0 ==
default value)
    int   maxCharWidth;                 // Max acceptable char width in pixels (0 ==
default value)
    int   minCharWidth;                 // Min acceptable char width in pixels (0 ==
default value)
```

```
    int   avgCharHeight;                // Average char height in pixels (0 == default
value)
    int   avgCharWidth;                 // Average char width in pixels (0 == default
value)

    int   xRoi[4];                      // ROI points - x coords
    int   yRoi[4];                      // ROI points - y coords

} JidoshaLightConfig;

//==============================================================================
//  JidoshaLightRecognition
//==============================================================================
typedef struct JidoshaLightRecognitionInfo
{
    double  totalTime;
    double  localizationTime;
    double  segmentationTime;
    double  classificationTime;
    double  loadDecodeTime;
    int     libVersion[3];
    char    libSHA1[41];

} JidoshaLightRecognitionInfo;

typedef struct JidoshaLightRecognition
{
    int frameId;                        // Unique Recognition ID
    char plate[8];                      // Plate text + byte 0 (null-terminated string)
    float probabilities[7];             // Range [0,1] - Recognition probability of each
character

    int xText;                          // Plate up-left corner X coord
    int yText;                          // Plate up-left corner Y coord
    int widthText;                      // Plate Width
    int heightText;                     // Plate Height

    int xChar[7];                       // Individual character up-left corner X coord
    int yChar[7];                       // Individual character up-left corner Y coord
    int widthChar[7];                   // Individual character width
    int heightChar[7];                  // Individual character height

    int textColor;                      // 0: dark text over bright background,
                                        // 1: bright text over dark background

    int isMotorcycle;                   // 0: false, 1: true
    int countryCode;                    // ISO 3166-1

    JidoshaLightRecognitionInfo info;   // Overall recognition benchmark information

} JidoshaLightRecognition;

//==============================================================================
//  JidoshaLightLicenseInfo
//==============================================================================
typedef struct JidoshaLightLicenseInfo
{
```

```
    uint64_t serial;
    char customer[64];
    int maxThreads;
    int maxConnections;
    int state;
    int ttl;
} JidoshaLightLicenseInfo;

//==============================================================================
//  JidoshaLightRecognitionList
//==============================================================================
typedef struct JidoshaLightRecognitionList JidoshaLightRecognitionList;
JL_API JidoshaLightRecognitionList* jidoshaLight_ANPR_createList();
JL_API JidoshaLightRecognitionList*
jidoshaLight_ANPR_duplicateList(JidoshaLightRecognitionList* list);
JL_API int jidoshaLight_ANPR_destroyList(JidoshaLightRecognitionList* list);
JL_API int jidoshaLight_ANPR_getListSize(JidoshaLightRecognitionList* list);
JL_API const JidoshaLightRecognition*
jidoshaLight_ANPR_getListElement(JidoshaLightRecognitionList* list, int pos);

//==============================================================================
//  JidoshaLightImage
//==============================================================================
typedef struct JidoshaLightImage JidoshaLightImage;
JL_API JidoshaLightImage* jidoshaLight_ANPR_createImage();
JL_API JidoshaLightImage* jidoshaLight_ANPR_duplicateImage(JidoshaLightImage* img);
JL_API int jidoshaLight_ANPR_destroyImage(JidoshaLightImage* img);
JL_API int jidoshaLight_ANPR_setImageLazyDecode(JidoshaLightImage* img, int enable);
JL_API int jidoshaLight_ANPR_loadImageFromFile(
    JidoshaLightImage* img,
    const char* filename
);
JL_API int jidoshaLight_ANPR_loadImageFromMemory(
    JidoshaLightImage* img,
    const uint8_t* buffer,
    int bufferSize
);
JL_API int jidoshaLight_ANPR_loadImageFromRawImgFmt(
    JidoshaLightImage* img,
    const uint8_t* buffer,
    int width,
    int height,
    int stride,
    JidoshaLightRawImgFmt fmt
);

//==============================================================================
//  Library Information
//==============================================================================
JL_API int jidoshaLight_getVersion(int* major, int* minor, int* release);
JL_API const char* jidoshaLight_getBuildSHA1();        // ASCII encoded SHA1
JL_API const char* jidoshaLight_getBuildFlags();       // ASCII encoded Build Flags
JL_API int jidoshaLight_getLicenseInfo(JidoshaLightLicenseInfo* info);
JL_API int jidoshaLight_isRemoteApi();

//==============================================================================
//  Utilities
```

```
//==============================================================================
JL_API const char* jidoshaLight_getReturnCodeString(int rc);
```

6.2.1.2 jidosha_light_api.h

```
//==============================================================================
//  PROCESSING
//==============================================================================
JL_API int jidoshaLight_ANPR_fromFile (
    const char* filename,
    JidoshaLightConfig* config,
    JidoshaLightRecognition* rec
);

JL_API int jidoshaLight_ANPR_fromMemory (
    const unsigned char* buffer,
    int bufferSize,
    JidoshaLightConfig* config,
    JidoshaLightRecognition* rec
);

JL_API int jidoshaLight_ANPR_fromLuma (
    unsigned char* luma,
    int width,
    int height,
    JidoshaLightConfig* config,
    JidoshaLightRecognition* rec
);

JL_API int jidoshaLight_ANPR_fromRawImgFmt (
    const unsigned char* buffer,
    int width,
    int height,
    int stride,
    JidoshaLightRawImgFmt fmt,
    JidoshaLightConfig* config,
    JidoshaLightRecognition* rec
);

JL_API int jidoshaLight_ANPR_fromImage(
    JidoshaLightImage* img,
    JidoshaLightConfig* config,
    JidoshaLightRecognition* rec
);

JL_API int jidoshaLight_ANPR_multi_fromImage (
    JidoshaLightImage* img,
    JidoshaLightConfig* config,
    int maxPlates,
    JidoshaLightRecognitionList* list
);
```

6.2.1.3 Tipos

enum JidoshaLightVehicleType	
Descrição	Define os tipos de placa que o OCR deve buscar na imagem.
Membros	JIDOSHA_LIGHT_VEHICLE_TYPE_CAR: apenas placas de carro JIDOSHA_LIGHT_VEHICLE_TYPE_MOTO: apenas placas de moto JIDOSHA_LIGHT_VEHICLE_TYPE_BOTH: ambos os tipos de placa

enum JidoshaLightMode	
Descrição	Define as estratégias de processamento que podem ser utilizadas pelo algoritmo de leitura de placas. A opção *FAST* utiliza o menor esforço computacional possível para a leitura enquanto a *ULTRA_SLOW* utiliza o maior. Quanto maior o nível de esforço maior a probabilidade de leitura da placa.
Membros	JIDOSHA_LIGHT_VEHICLE_TYPE_CAR: apenas placas de carro JIDOSHA_LIGHT_MODE_DISABLE: valor reservado para uso futuro. Atualmente possui o mesmo efeito da opção ULTRA_SLOW JIDOSHA_LIGHT_MODE_FAST: estratégia mais rápida de processamento, seu uso é aconselhado para casos onde o tempo de processamento é crítico JIDOSHA_LIGHT_MODE_NORMAL: estratégia de processamento moderada, possui tempo de processamento e índice de reconhecimento superiores ao método FAST JIDOSHA_LIGHT_MODE_SLOW: estratégia de processamento lenta, possui tempo de processamento e índice de reconhecimento superiores ao método NORMAL JIDOSHA_LIGHT_MODE_ULTRA_SLOW: estratégia de processamento super lenta, possui o maior tempo de processamento e o maior índice de reconhecimento Um dos modos acima necessariamente deve ser usado. Além disso, opcionalmente pode-se selecionar a estratégia de localização utilizada pelo algoritmo de leitura de placas. A opção LOCALIZATION_MODE_0 é a padrão. As outras opções atualmente afetam apenas o processamento de placas do Brasil. JIDOSHA_LIGHT_LOCALIZATION_MODE_0: estratégia de localização com maior tempo de processamento e maior índice de reconhecimento JIDOSHA_LIGHT_LOCALIZATION_MODE_1: estratégia de localização um pouco mais rápida e com índice de reconhecimento um pouco menor JIDOSHA_LIGHT_LOCALIZATION_MODE_2: estratégia de localização rápida, com índice de reconhecimento menor, aplicável apenas a placas de carros, não de motos

enum JidoshaLightCountryCode	
Descrição	Define o código dos países suportados pela biblioteca de reconhecimento no formato ISO 3166-1. O uso de um determinado país é limitado pela licença.
Membros	JIDOSHA_LIGHT_COUNTRY_CODE_CONESUL JIDOSHA_LIGHT_COUNTRY_CODE_ARGENTINA JIDOSHA_LIGHT_COUNTRY_CODE_BRAZIL JIDOSHA_LIGHT_COUNTRY_CODE_CHILE JIDOSHA_LIGHT_COUNTRY_CODE_COLOMBIA JIDOSHA_LIGHT_COUNTRY_CODE_MEXICO

	JIDOSHA_LIGHT_COUNTRY_CODE_PARAGUAY JIDOSHA_LIGHT_COUNTRY_CODE_URUGUAY JIDOSHA_LIGHT_COUNTRY_CODE_NETHERLANDS JIDOSHA_LIGHT_COUNTRY_CODE_FRANCE

enum JidoshaLightRawImgFmt	
Descrição	Define os tipos de formato RAW suportados pela biblioteca.
Membros	JIDOSHA_LIGHT_IMG_FMT_XRGB_8888: formato XRGB 32 bits, sendo o byte menos significativo utilizado para o canal azul (*Blue*). O byte mais significativo é ignorado JIDOSHA_LIGHT_IMG_FMT_RGB_888: formato RGB 24 bits, sendo o byte menos significativo utilizado para o canal azul (*Blue*) JIDOSHA_LIGHT_IMG_FMT_LUMA: formato de 8 bits contendo apenas o canal de luminância JIDOSHA_LIGHT_IMG_FMT_YUV420: formato YUV 8 bits não entrelaçado com subsampling 4:2:0

struct JidoshaLightConfig	
Descrição	Define os tipos de formato RAW suportados pela biblioteca.
Membros	*int configId*: campo reservado para uso futuro, tem seu valor ignorado pela biblioteca *int vehicleType*: indica o tipo de placa que o OCR deve buscar. Ver enum JidoshaLightVehicleType *int processingMode*: indica a estratégia de processamento a ser utilizada. Ver enum JidoshaLightMode *int timeout*: indica o tempo máximo em milissegundos para o reconhecimento de placa. O valor `0` indica que não há timeout. Um valor diferente de `0` ajuda a manter baixo o tempo médio de processamento - o processamento é interrompido e a função retorna assim que o timeout expira. O valor deve ser determinado com base na resolução da imagem e CPU utilizada *int countryCode*: indica o código do país cuja placa se pretende reconhecer. Ver enum JidoshaLightCountryCode *float minProbPerChar*: valor de `*0.0f*` a `*1.0f*` usado para definir a probabilidade mínima que um determinado caractere da placa deve ter para ser considerado válido **(recomendado: 0.85)** *int maxLowProbabilityChars*: número máximo de caracteres com probabilidade menor a `*minProbPerChar*` para que a placa reconhecida seja considerada válida - uma placa reconhecida como inválida é retornada como uma string vazia '\0' *char lowProbabilityChar*: caractere que será utilizado para substituir aqueles com probabilidade inferior a `*minProbPerChar*` *float avgPlateAngle*: ângulo médio das placas nas imagens em relação ao eixo horizontal *float avgPlateSlant*: ângulo médio da inclinação dos caracteres nas imagens em relação ao eixo vertical *int maxCharHeight*: altura máxima aceitável dos caracteres, em pixels *int minCharHeight*: altura mínima aceitável dos caracteres, em pixels *int maxCharWidth*: largura máxima aceitável dos caracteres, em pixels *int minCharWidth*: largura mínima aceitável dos caracteres, em pixels *int avgCharHeight*: altura média dos caracteres, em pixels (valor padrão: 20)

	int avgCharWidth: largura média dos caracteres, em pixels (valor padrão: 7) *int xRoi[4]* e *int yRoi[4]*: coordenadas x e y dos quatro pontos da região de interesse da imagem (ROI - Region Of Interest) em qualquer ordem. Compreende-se por região de interesse um quadrilátero dentro da imagem onde espera-se encontrar as placas a serem reconhecidas. O uso da ROI beneficia o tempo de processamento e a taxa de acertos, uma vez que exclui regiões de acostamento ou locais sem importância para o processo de reconhecimento de placas. Definindo todas as coordenadas iguais a zero fará com que a ROI seja ignorada e toda a imagem seja processada. Valores maiores que as dimensões da imagem ou negativos resultam no retorno do código de erro JIDOSHA_LIGHT_ERROR_INVALID_ROI. A mudança destes valores causa o recálculo da região da ROI, impactando no tempo de processamento da primeira imagem após a alteração.

Atenção: As coordenadas dos pontos da ROI têm sua origem (0,0) no canto superior esquerdo da imagem e se estendem até o canto inferior direito (largura-1, altura-1). Assim para uma imagem de resolução 800x600, os valores válidos para os pontos da ROI vão de (0,0) até (799,599). Cabe ressaltar ainda que os 4 pontos não podem ser colineares.

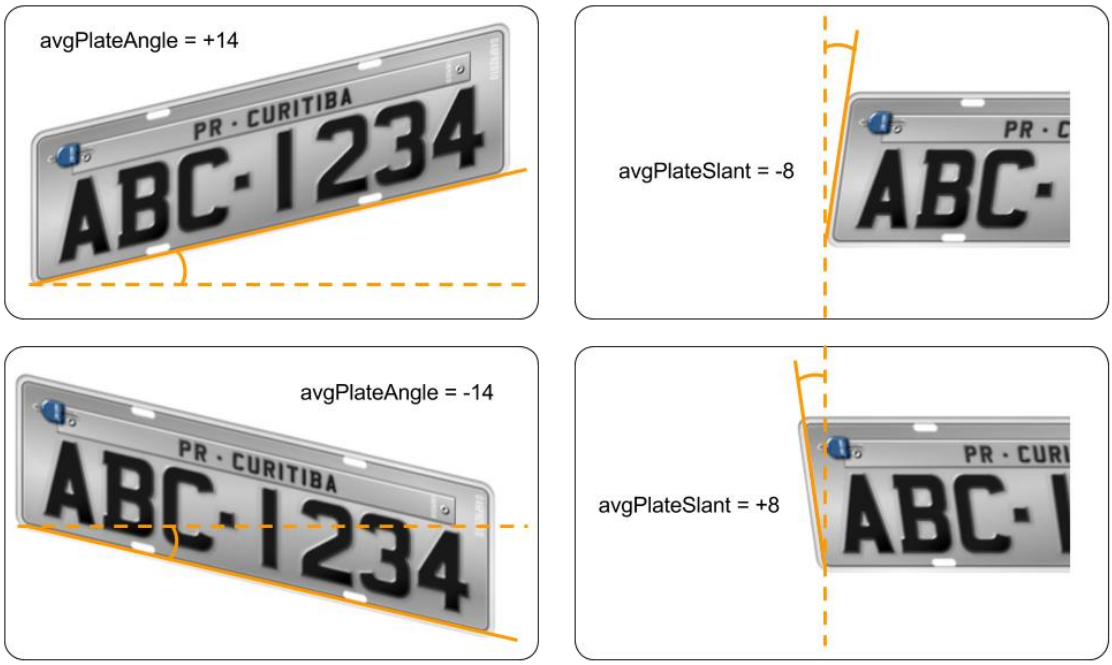


Figura 10 - Como calcular os valores de avgPlateAngle e avgPlateSlant

struct JidoshaLightRecognitionInfo	
Descrição	A finalidade dessa estrutura é trazer informações relativas ao tempo de processamento do *JidoshaLight*, facilitando o diagnóstico de desempenho. Todos os tempos são fornecidos em milissegundos.
Membros	*double totalTime*: tempo total de processamento da imagem (somatório dos demais tempos). *double localizationTime*: tempo gasto na etapa de localização da placa na imagem. *double segmentationTime*: tempo gasto na etapa de extração dos caracteres da placa. *double classificationTime*: tempo gasto na etapa de classificação dos caracteres da placa. *double loadDecodeTime*: tempo gasto na leitura e na decodificação do arquivo da imagem. *int libVersion[3]*: versão da biblioteca que processou a imagem. *char libSHA1[41]*: identificador da biblioteca que processou a imagem.

struct JidoshaLightRecognition	
Descrição	A finalidade dessa estrutura é guardar o resultado do reconhecimento de placa, incluindo: os caracteres da placa, a confiabilidade de cada caractere, e as coordenadas da placa na imagem.
Membros	*int frameId*: campo reservado para uso futuro, seu valor é sempre zerado. *char plate[8]*: placa de 7 caracteres terminada com 0, ou string vazia se a placa não foi encontrada. *float probabilities[7]*: valores de 0.0 a 1.0 indicando a confiabilidade, na forma de probabilidade, do reconhecimento de cada caractere. *int xText* e *int yText*: coordenadas do canto superior esquerdo da placa, caso tenha sido encontrada. *int widthText*: largura do retângulo da placa. *int heightText*: altura do retângulo da placa. *int xChar[7]* e *int yChar[7]*: coordenada do canto superior esquerdo de cada um dos caracteres reconhecidos. *int widthChar[7]*: largura do retângulo de cada um dos caracteres reconhecidos. *int heightChar[7]*: altura do retângulo de cada um dos caracteres reconhecidos. *int textColor*: cor do texto da placa, 0 - escuro, 1 - claro. *int isMotorcycle*: indica se placa é de moto, 0 - não-moto, 1 - moto. *int countryCode*: indica o código do país (no padrão ISO 3166-1) da placa reconhecida. Os possíveis valores para este campo estão definidos na enum JidoshaLightCountryCode. *JidoshaLightRecognitionInfo info*: estrutura contendo informações relativas ao tempo de processamento.

struct JidoshaLightLicenseInfo	
Descrição	Struct utilizada para armazenar as informações sobre a licença utilizada pela biblioteca JidoshaLight.
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 Códigos de retorno de função) *int ttl*: time-to-live em horas para licenças do tipo RTC. Este campo possui o valor **-1** caso a licença não seja expirável

struct JidoshaLightRecognitionList	
Descrição	Tipo opaco utilizado pela biblioteca para retornar uma lista de objetos do tipo JidoshaLightRecognition. Funções que manipulam a lista inserem novos elementos ao final dela. Para limpar uma lista, basta destruir e criar uma nova. Para evitar vazamentos de memória, o usuário deve sempre destruir as listas que não estão mais utilizadas. Este tipo não é thread safe.
Membros	Nenhum

Métodos relacionados	jidoshaLight_ANPR_createList jidoshaLight_ANPR_duplicateList jidoshaLight_ANPR_destroyList jidoshaLight_ANPR_getListSize jidoshaLight_ANPR_getListElement

struct JidoshaLightImage	
Descrição	Tipo opaco utilizado pela biblioteca para carregar uma imagem a ser processada. Depois de criado, um objeto *JidoshaLightImage* pode ser utilizado para carregar inúmeras imagens, mesmo que sejam de formatos diferentes. Entretanto, chamadas subsequentes causam a substituição do conteúdo previamente carregado. O processo de decode da imagem pode ser postergado para o momento do processamento caso o modo *LazyDecode* esteja habilitado. Nesta situação, as funções de *load* apenas armazenam o conteúdo do buffer raw da imagem sem efetuar processamento algum. Este comportamento é útil para aplicações cliente-servidor, já que o custo computacional do decode é delegado ao servidor. Para evitar vazamentos de memória o usuário deve sempre destruir as imagens que não estão mais sendo utilizadas. Este tipo não é thread safe.
Membros	Nenhum
Métodos relacionados	jidoshaLight_ANPR_createImage jidoshaLight_ANPR_duplicateImage jidoshaLight_ANPR_destroyImage jidoshaLight_ANPR_setImageLazyDecode jidoshaLight_ANPR_loadImageFromFile jidoshaLight_ANPR_loadImageFromMemory jidoshaLight_ANPR_loadImageFromRawImgFmt

6.2.1.4 Métodos

jidoshaLight_ANPR_createList	
Protótipo da Função	`JidoshaLightRecognitionList* jidoshaLight_ANPR_createList();`
Descrição	Função utilizada para criar uma JidoshaLightRecognitionList vazia
Parâmetros	Nenhum
Retorno	Um ponteiro válido para o tipo *JidoshaLightRecognitionList* ou *NULL* em caso de falha.

jidoshaLight_ANPR_duplicateList	
Protótipo da Função	`JidoshaLightRecognitionList* jidoshaLight_ANPR_duplicateList(JidoshaLightRecognitionList* list);`
Descrição	Função utilizada para duplicar uma JidoshaLightRecognitionList
Parâmetros	*JidoshaLightRecognitionList* list*: ponteiro para um objeto *JidoshaLightRecognitionList*

Retorno	Um ponteiro válido para o tipo *JidoshaLightRecognitionList* ou *NULL* em caso de falha.

jidoshaLight_ANPR_destroyList	
Protótipo da Função	`int jidoshaLight_ANPR_destroyList(JidoshaLightRecognitionList* list);`
Descrição	Função utilizada para destruir os objetos criados pelas funções *jidoshaLight_ANPR_createList* e *jidoshaLight_ANPR_duplicateList*
Parâmetros	*JidoshaLightRecognitionList* list*: ponteiro válido para um objeto *JidoshaLightRecognitionList*
Retorno	Código de retorno *JIDOSHA_LIGHT_SUCCESS* no caso de sucesso, outro código caso contrário (ver Códigos de retorno de função)

jidoshaLight_ANPR_getListSize	
Protótipo da Função	`int jidoshaLight_ANPR_getListSize(JidoshaLightRecognitionList* list);`
Descrição	Função utilizada para ler a quantidade de elementos armazenados dentro da lista
Parâmetros	*JidoshaLightRecognitionList* list*: ponteiro válido para um objeto *JidoshaLightRecognitionList*
Retorno	Número de elementos presentes na lista (valor maior ou igual a zero) ou -1 em caso de erro (**list* inválido).

jidoshaLight_ANPR_getListElement	
Protótipo da Função	`const JidoshaLightRecognition* jidoshaLight_ANPR_getListElement(JidoshaLightRecognitionList* list, int pos);`
Descrição	Função utilizada para recuperar um ponteiro para o elemento presente na posição *pos* da lista. O conteúdo do ponteiro retornado não pode ser modificado pelo usuário (const).
Parâmetros	*JidoshaLightRecognitionList* list*: ponteiro válido para um objeto *JidoshaLightRecognitionList* *int pos*: posição do elemento a ser recuperado no intervalo ` *[0,ListSize)*
Retorno	Ponteiro válido para um objeto imutável do tipo JidoshaLightRecognition ou *NULL* em caso de erro (**list* inválido ou *pos* fora do intervalo).

jidoshaLight_ANPR_createImage	
Protótipo da Função	`JidoshaLightImage* jidoshaLight_ANPR_createImage();`
Descrição	Função utilizada para criar uma JidoshaLightImage
Parâmetros	Nenhum
Retorno	Ponteiro válido para o tipo *JidoshaLightImage* ou *NULL* em caso de falha.

jidoshaLight_ANPR_duplicateImage	
Protótipo da Função	`JidoshaLightImage* jidoshaLight_ANPR_duplicateImage(JidoshaLightImage* img);`
Descrição	Função utilizada para criar uma JidoshaLightImage. A imagem duplicada herda o estado da imagem original.
Parâmetros	*JidoshaLightImage* img*: ponteiro para um objeto *JidoshaLightImage*
Retorno	Ponteiro válido para o tipo *JidoshaLightImage* ou *NULL* em caso de falha.

jidoshaLight_ANPR_destroyImage	
Protótipo da Função	`int jidoshaLight_ANPR_destroyImage(JidoshaLightImage* img);`
Descrição	Função utilizada para destruir os objetos criados pelas funções jidoshaLight_ANPR_createImage e jidoshaLight_ANPR_duplicateImage
Parâmetros	*JidoshaLightImage* img*: ponteiro para um objeto *JidoshaLightImage*
Retorno	Código de retorno *JIDOSHA_LIGHT_SUCCESS* no caso de sucesso, outro código caso contrário (ver Códigos de retorno de função)

jidoshaLight_ANPR_setImageLazyDecode	
Protótipo da Função	`int jidoshaLight_ANPR_setImageLazyDecode(JidoshaLightImage* img, int enable);`
Descrição	Função utilizada para habilitar o modo *LazyDecode* (ver descrição em JidoshaLightImage). A mudança tem efeito imediato e invalida qualquer imagem anteriormente carregada.
Parâmetros	*JidoshaLightImage* img*: ponteiro para um objeto *JidoshaLightImage* *int enable*: 0 desabilitado (default), 1 habilitado
Retorno	Código de retorno *JIDOSHA_LIGHT_SUCCESS* no caso de sucesso, outro código caso contrário (ver Códigos de retorno de função)

jidoshaLight_ANPR_loadImageFromFile	

Protótipo da Função	`int jidoshaLight_ANPR_loadImageFromFile (`
	`  JidoshaLightImage* img,`
	`  const char* filename`
	`);`

Descrição	Função utilizada para carregar uma *JidoshaLightImage* a partir de um arquivo. Formatos de arquivo suportados: JPEG, BMP, PNG e TIFF.
Parâmetros	*JidoshaLightImage* img*: ponteiro para um objeto *JidoshaLightImage* *const char* filename*: caminho absoluto para o arquivo a ser carregado
Retorno	Código de retorno *JIDOSHA_LIGHT_SUCCESS* no caso de sucesso, outro código caso contrário (ver Códigos de retorno de função)

jidoshaLight_ANPR_loadImageFromMemory	

Protótipo da Função	`int jidoshaLight_ANPR_loadImageFromMemory (`
	`  JidoshaLightImage* img,`
	`  const uint8_t* buffer,`
	`  int bufferSize`
	`);`

Descrição	Função utilizada para carregar uma *JidoshaLightImage* a partir de um arquivo já carregado na memória. Formatos de arquivo suportados: JPEG, BMP, PNG e TIFF.
Parâmetros	*JidoshaLightImage* img*: ponteiro para um objeto *JidoshaLightImage* *const uint8_t* buffer*: ponteiro para o buffer contendo o arquivo carregado *int bufferSize*: tamanho em bytes do buffer
Retorno	Código de retorno *JIDOSHA_LIGHT_SUCCESS* no caso de sucesso, outro código caso contrário (ver Códigos de retorno de função)

jidoshaLight_ANPR_loadImageFromRawImgFmt	

Protótipo da Função	`int jidoshaLight_ANPR_loadImageFromRawImgFmt (`
	`  JidoshaLightImage* img,`
	`  const uint8_t* buffer,`
	`  int width,`
	`  int height,`
	`  int stride,`
	`  JidoshaLightRawImgFmt fmt`
	`);`

Descrição	Função utilizada para carregar uma *JidoshaLightImage* a partir de um buffer contendo uma imagem no formato RAW. Ver formatos suportados em enum JidoshaLightRawImgFmt
Parâmetros	*JidoshaLightImage* img*: ponteiro válido para um objeto *JidoshaLightImage* *const uint8_t* buffer*: ponteiro para o buffer contendo a imagem em formato RAW *int width*: largura em pixels da imagem *int height*: altura em pixels da imagem *int stride*: tamanho em bytes de uma linha da imagem *JidoshaLightRawImgFmt fmt*: formato da imagem
Retorno	Código de retorno *JIDOSHA_LIGHT_SUCCESS* no caso de sucesso, outro código caso contrário (ver Códigos de retorno de função)

jidoshaLight_ANPR_fromFile	

Protótipo da Função	`int jidoshaLight_ANPR_fromFile (`
	`  const char* filename,`
	`  JidoshaLightConfig* config,`
	`  JidoshaLightRecognition* rec`
	`);`

Descrição	Reconhece uma placa a partir de um arquivo de imagem cujo caminho é fornecido em *const char* filename*. Utiliza as configurações definidas em *JidoshaLightConfig* config* e retorna o resultado do reconhecimento na struct *JidoshaLightRecognition* rec*. Se não for possível encontrar uma placa na imagem, o campo *rec->plate* é retornado vazio.

	Caso ocorra algum erro durante o processamento, a struct *JidoshaLightRecognition* rec* será retornada vazia e um valor diferente de *JIDOSHA_LIGHT_SUCCESS* será retornado pela função. Os possíveis valores de retorno estão definidos na *enum JidoshaLightReturnCode*. Ver formatos suportados em jidoshaLight_ANPR_loadImageFromFile.
Parâmetros	*const char* filename*: caminho para o arquivo da imagem. *JidoshaLightConfig* config*: ponteiro para a `struct JidoshaLightConfig` com a configuração para a biblioteca. Um ponteiro `NULL` neste parâmetro causa o uso das configurações padrão da biblioteca. *JidoshaLightRecognition* rec*: ponteiro para a `struct JidoshaLightRecognition` onde será armazenado o resultado da leitura.
Retorno	Código de retorno *JIDOSHA_LIGHT_SUCCESS* no caso de sucesso, outro código caso contrário (ver Códigos de retorno de função)

jidoshaLight_ANPR_fromMemory	

Protótipo da Função	`int jidoshaLight_ANPR_fromMemory (`
	`  const unsigned char* buffer,`
	`  int bufferSize,`
	`  JidoshaLightConfig* config,`
	`  JidoshaLightRecognition* rec`
	`);`

Descrição	Reconhece uma placa a partir de um *buffer* contendo um arquivo de imagem previamente carregado na memória. Utiliza as configurações definidas em *JidoshaLightConfig* config* e retorna o resultado do reconhecimento na struct *JidoshaLightRecognition* rec*. Se não for possível encontrar uma placa na imagem, o campo *rec->plate* é retornado vazio. Caso ocorra algum erro durante o processamento, a struct *JidoshaLightRecognition* rec* será retornada vazia e um valor diferente de *JIDOSHA_LIGHT_SUCCESS* será retornado pela função. Os possíveis valores de retorno estão definidos na *enum JidoshaLightReturnCode*. Ver formatos suportados em jidoshaLight_ANPR_loadImageFromMemory.
Parâmetros	*const unsigned char* buffer*: array de bytes que contém a imagem. *int bufferSize*: tamanho do array de bytes. *JidoshaLightConfig* config*: ponteiro para a *struct JidoshaLightConfig* com a configuração para a biblioteca. Um ponteiro *NULL* neste parâmetro causa o uso das configurações padrão da biblioteca. *JidoshaLightRecognition* rec*: ponteiro para a *struct JidoshaLightRecognition* onde será armazenado o resultado da leitura.
Retorno	Código de retorno *JIDOSHA_LIGHT_SUCCESS* no caso de sucesso, outro código caso contrário (ver Códigos de retorno de função)

jidoshaLight_ANPR_fromLuma	

Protótipo da Função	`int jidoshaLight_ANPR_fromLuma (`
	`  unsigned char* luma,`
	`  int width,`
	`  int height,`
	`  JidoshaLightConfig* config,`
	`  JidoshaLightRecognition* rec`
	`);`

Descrição	Reconhece uma placa a partir de um *buffer* contendo uma imagem no formato RAW grayscale 8-bits. Utiliza as configurações definidas em *JidoshaLightConfig* config* e retorna o resultado do reconhecimento na struct *JidoshaLightRecognition* rec*. Se não for possível encontrar uma placa na imagem, o campo *rec->plate* é retornado vazio. Caso ocorra algum erro durante o processamento, a struct *JidoshaLightRecognition* rec* será retornada vazia e um valor diferente de *JIDOSHA_LIGHT_SUCCESS* será retornado pela função. Os possíveis valores de retorno estão definidos na *enum JidoshaLightReturnCode*.
Parâmetros	*unsigned char* luma*: array de bytes que contém a imagem no formato RAW grayscale 8-bits. *int width*: largura da imagem. *int height*: altura da imagem. *JidoshaLightConfig* config*: ponteiro para a *struct JidoshaLightConfig* com a configuração para a biblioteca. Um ponteiro *NULL* neste parâmetro causa o uso das configurações padrão da biblioteca. *JidoshaLightRecognition* rec*: ponteiro para a *struct JidoshaLightRecognition* onde será armazenado o resultado da leitura.
Retorno	Código de retorno *JIDOSHA_LIGHT_SUCCESS* no caso de sucesso, outro código caso contrário (ver Códigos de retorno de função)

jidoshaLight_ANPR_fromRawImgFmt	

Protótipo da Função	`int jidoshaLight_ANPR_fromRawImgFmt (`
	`  const unsigned char* buffer,`
	`  int width,`
	`  int height,`
	`  int stride,`
	`  JidoshaLightRawImgFmt fmt,`
	`  JidoshaLightConfig* config,`
	`  JidoshaLightRecognition* rec`
	`);`

Descrição	Reconhece uma placa a partir de um *buffer* contendo uma imagem em algum dos formatos RAW definidos na *enum JidoshaLightRawImgFmt*. Utiliza as configurações definidas em *JidoshaLightConfig* config* e retorna o resultado do reconhecimento na struct *JidoshaLightRecognition* rec*. Se não for possível encontrar uma placa na imagem, o campo *rec->plate* é retornado vazio. Caso ocorra algum erro durante o processamento, a struct *JidoshaLightRecognition* rec* será retornada vazia e um valor diferente de *JIDOSHA_LIGHT_SUCCESS* será retornado pela função. Os possíveis valores de retorno estão definidos na *enum JidoshaLightReturnCode*. Ver formatos suportados em jidoshaLight_ANPR_loadImageFromRawImgFmt.
Parâmetros	*const unsigned char* buffer*: array de bytes contendo a imagem no formato RAW. *int width*: largura da imagem. *int height*: altura da imagem. *int stride*: tamanho em bytes de cada linha da imagem. *JidoshaLightRawImgFmt fmt*: formato da imagem. *JidoshaLightConfig* config*: ponteiro para a *struct JidoshaLightConfig* com a configuração para a biblioteca. Um ponteiro *NULL* neste parâmetro causa o uso das configurações padrão da biblioteca. *JidoshaLightRecognition* rec*: ponteiro para a *struct JidoshaLightRecognition* onde será armazenado o resultado da leitura.

Retorno	Código de retorno *JIDOSHA_LIGHT_SUCCESS* no caso de sucesso, outro código caso contrário (ver Códigos de retorno de função)

jidoshaLight_ANPR_fromImage	

Protótipo da Função	`int jidoshaLight_ANPR_fromImage(`
	`  JidoshaLightImage* img,`
	`  JidoshaLightConfig* config,`
	`  JidoshaLightRecognition* rec`
	`);`

Descrição	Reconhece uma placa a partir de uma JidoshaLightImage previamente carregada. Utiliza as configurações definidas em *JidoshaLightConfig* config* e retorna o resultado do reconhecimento na struct *JidoshaLightRecognition* rec*. Se não for possível encontrar uma placa na imagem, o campo *rec->plate* é retornado vazio. Caso ocorra algum erro durante o processamento, a struct *JidoshaLightRecognition* rec* será retornada vazia e um valor diferente de *JIDOSHA_LIGHT_SUCCESS* será retornado pela função. Os possíveis valores de retorno estão definidos na *enum JidoshaLightReturnCode*.
Parâmetros	*JidoshaLightImage* img*: ponteiro para uma JidoshaLightImage válida *JidoshaLightConfig* config*: ponteiro para a *struct JidoshaLightConfig* com a configuração para a biblioteca. Um ponteiro *NULL* neste parâmetro causa o uso das configurações padrão da biblioteca. *JidoshaLightRecognition* rec*: ponteiro para a *struct JidoshaLightRecognition* onde será armazenado o resultado da leitura.
Retorno	Código de retorno *JIDOSHA_LIGHT_SUCCESS* no caso de sucesso, outro código caso contrário (ver Códigos de retorno de função)

jidoshaLight_ANPR_multi_fromImage	

Protótipo da Função	`int jidoshaLight_ANPR_multi_fromImage (`
	`  JidoshaLightImage* img,`
	`  JidoshaLightConfig* config,`
	`  int maxPlates,`
	`  JidoshaLightRecognitionList* list`
	`);`

Descrição	Reconhece múltiplas placas a partir de uma JidoshaLightImage previamente carregada. Utiliza as configurações definidas em *JidoshaLightConfig* config* adiciona `maxPlates` reconhecimentos ao final da `*JidoshaLightRecognitionList\* list*`. Caso o número de placas encontradas seja menor que o valor especificado, elementos `*JidoshaLightRecognition*` vazios serão adicionados à lista até preencher `*maxPlates*`. Caso ocorra algum erro elementos *JidoshaLightRecognition* vazios serão adicionados à lista e um código de retorno diferente de *JIDOSHA_LIGHT_SUCCESS* será retornado pela função (ver enum JidoshaLightReturnCode).
Parâmetros	*JidoshaLightImage* img*: ponteiro para uma JidoshaLightImage válida *JidoshaLightConfig* config*: ponteiro para a *struct JidoshaLightConfig* com a configuração para a biblioteca. Um ponteiro *NULL* neste parâmetro causa o uso das configurações padrão da biblioteca. *int maxPlates:* número máximo de placas a serem reconhecidas (1 ou mais)

	JidoshaLightRecognitionList list*: ponteiro para um objeto JidoshaLightRecognitionList onde serão adicionados *maxPlates* novos JidoshaLightRecognition.
Retorno	Código de retorno *JIDOSHA_LIGHT_SUCCESS* no caso de sucesso, outro código caso contrário (ver Códigos de retorno de função)

jidoshaLight_getVersion	
Protótipo da Função	`int jidoshaLight_getVersion(int* major, int* minor, int* release);`
Descrição	Usada para verificar a versão da biblioteca, no formato major.minor.release.
Parâmetros	*int major, minor, release*: ponteiros para variáveis int onde serão escritos os números que compõem a versão.
Retorno	Sempre retorna *JIDOSHA_LIGHT_SUCCESS*.

jidoshaLight_getBuildSHA1	
Protótipo da Função	`const char* jidoshaLight_getBuildSHA1();`
Descrição	Usada para verificar o SHA1 do build da biblioteca.
Parâmetros	Nenhum
Retorno	Retorna um ponteiro para o início de uma string terminada em *|0* contendo o valor do SHA1 do build.

jidoshaLight_getBuildFlags	
Protótipo da Função	`const char* jidoshaLight_getBuildFlags();`
Descrição	Usada para verificar as opções do build da biblioteca.
Parâmetros	Nenhum
Retorno	Retorna um ponteiro para o início de uma string terminada em *|0* contendo as opções do build.

jidoshaLight_isRemoteApi	
Protótipo da Função	`JL_API int jidoshaLight_isRemoteApi();`
Descrição	Verifica se a biblioteca da aplicação implementa processamento local ou remoto.
Parâmetros	Nenhum

Retorno	Retorna *0* caso esteja implementada API com acesso local ou diferente deste valor caso o processamento seja remoto.

jidoshaLight_getLicenseInfo	
Protótipo da Função	`int jidoshaLight_getLicenseInfo(JidoshaLightLicenseInfo* info)`
Descrição	Função utilizada para ler as informações da licença utilizada pela biblioteca JidoshaLight.
Parâmetros	*JidoshaLightLicenseInfo* info*: ponteiro para uma struct JidoshaLightLicenseInfo
Retorno	Retorna *JIDOSHA_LIGHT_SUCCESS* em caso de sucesso.

jidoshaLight_getReturnCodeString	
Protótipo da Função	`const char* jidoshaLight_getReturnCodeString(int rc)`
Descrição	Função utilizada para converter um código de retorno da biblioteca em uma string C.
Parâmetros	*int rc*: algum código de retorno definido em enum JidoshaLightReturnCode e enum JidoshaLightReturnCodeNetwork
Retorno	String representativa do código de erro.

6.2.1.5 Códigos de retorno de função

Os códigos de retorno das funções são relacionados ao processo de reconhecimento (*enum JidoshaLightReturnCode*`) ou ao processo de comunicação remota (`*enum JidoshaLightReturnCodeNetwork*`). Códigos:

- JIDOSHA_LIGHT_ERROR_FILE_NOT_FOUND: retornado pelas funções jidoshaLight_ANPR_fromFile e jidoshaLight_ANPR_loadImageFromFile quando o caminho do arquivo especificado não existe.
- JIDOSHA_LIGHT_ERROR_INVALID_IMAGE: retornado pelas funções de processamento e carga de imagens. Ocorre quando a imagem passada está corrompida.
- JIDOSHA_LIGHT_ERROR_INVALID_IMAGE_TYPE: retornado pelas funções jidoshaLight_ANPR_fromFile, jidoshaLight_ANPR_fromMemory e funções relacionadas a carga da JidoshaLightImage. Ocorre quando se tenta processar uma imagem de formato não suportado.
- JIDOSHA_LIGHT_ERROR_INVALID_IMAGE_SIZE: retornado pelas funções jidoshaLight_ANPR_fromFile, jidoshaLight_ANPR_fromMemory e funções relacionadas a carga da JidoshaLightImage. Ocorre quando se tenta processar uma imagem cujo tamanho excede os límites máximos suportados pela biblioteca (ARM Zynq: 1280x960px, Outros: 2500x2500px).
- JIDOSHA_LIGHT_ERROR_INVALID_PROPERTY: retornado por todas as funções que possuem argumentos. Ocorre quando o argumento é inválido. No caso de funções que recebem ponteiros, este código é retornado quando o argumento é **NULL** (exceto nos casos em que **NULL** é um valor válido para o argumento).
- JIDOSHA_LIGHT_ERROR_COUNTRY_NOT_SUPPORTED: retornado pelas funções **ANPR** quando o código do país fornecido na estrutura de configuração não é suportado pela biblioteca.
- JIDOSHA_LIGHT_ERROR_API_CALL_NOT_SUPPORTED: retornado quando uma função da API não está disponível para uma determinada plataforma.

- JIDOSHA_LIGHT_ERROR_INVALID_ROI: retornado quando uma região de interesse inválida é fornecida. Ver a descrição da struct JidoshaLightConfig para maiores informações.
- JIDOSHA_LIGHT_ERROR_INVALID_*HANDLE*: retornado quando o *handle* passado para a função não foi inicializado corretamente.
- JIDOSHA_LIGHT_ERROR_API_CALL_HAS_NO_EFFECT: retornado quando uma função da API não teve efeito ao ser executada. Pode ocorrer quando existe precedência entre chamadas.
- JIDOSHA_LIGHT_ERROR_LICENSE_INVALID: retornado pelas funções **ANPR** quando o *hardkey* não está presente ou apresenta problemas. Contate a Pumatronix Equipamentos Eletrônicos para maiores informações.
- JIDOSHA_LIGHT_ERROR_LICENSE_EXPIRED: retornado pelas funções **ANPR** quando um *hardkey* do tipo demonstração expirou. Contate a Pumatronix Equipamentos Eletrônicos para maiores informações.
- JIDOSHA_LIGHT_ERROR_LICENSE_MAX_THREADS_EXCEEDED: retornado pelas funções **ANPR** quando o número máximo de threads concorrentes ultrapassa o permitido pela licença.
- JIDOSHA_LIGHT_ERROR_LICENSE_UNTRUSTED_RTC: retornado pelas funções **ANPR** quando uma licença com data limite de uso não tem disponível uma referência confiável de tempo/data.
- JIDOSHA_LIGHT_ERROR_OTHER: retornado quando um erro inesperado ocorre. Contate a Pumatronix Equipamentos Eletrônicos para suporte.
- JIDOSHA_LIGHT_ERROR_SERVER_CONNECT_FAILED: retornado quando uma chamada de API remota não consegue se conectar ao servidor.
- JIDOSHA_LIGHT_ERROR_SERVER_DISCONNECTED: retornado quando uma sessão remota com o servidor foi fechada inesperadamente.
- JIDOSHA_LIGHT_ERROR_SERVER_QUEUE_TIMEOUT: retornado quando uma requisição foi decartada no servidor por timeout.
- JIDOSHA_LIGHT_ERROR_SERVER_QUEUE_FULL: retornado quando uma requisição foi descartada no servidor por falta de espaço na fila.
- JIDOSHA_LIGHT_ERROR_SOCKET_IO_ERROR: retornado quando ocorreu um erro de IO de rede em uma sessão remota com o servidor.
- JIDOSHA_LIGHT_ERROR_SOCKET_WRITE_FAILED: retornado quando ocorre erro no envio de mensagens entre cliente e servidor remoto.
- JIDOSHA_LIGHT_ERROR_SOCKET_READ_TIMEOUT: retornado quando ocorre erro no recebimento de mensagens entre cliente e servidor remoto.
- JIDOSHA_LIGHT_ERROR_SOCKET_INVALID_RESPONSE: retornado quando uma mensagem inválida foi recebida.
- JIDOSHA_LIGHT_ERROR_*HANDLE*_QUEUE_FULL: retornado quando a fila de requisições pendentes atingiu o máximo para um determinado *handle* assíncrono.
- JIDOSHA_LIGHT_ERROR_SERVER_CONN_LIMIT_REACHED: retornado ao tentar conectar-se a um servidor com o número máximo de sessões abertas.
- JIDOSHA_LIGHT_ERROR_SERVER_VERSION_NOT_SUPPORTED: retornado quando a versão do servidor não é compatível com a versão da biblioteca do cliente.
- JIDOSHA_LIGHT_ERROR_SERVER_NOT_READY: retornado quando o servidor está iniciando mais ainda não está pronto para processar imagens. O cliente deve esperar e tentar reconectar.

6.2.2 API JidoshaLight C/C++ (Remota Síncrona)

A API Remota Síncrona estende a API Local, permitindo configurar um servidor remoto para processar as imagens remotamente ao invés de localmente. Deve ser utilizada em conjunto com a biblioteca *libjidoshaLightRemote.so*.

As chamadas *jidoshaLight_ANPR* definidas na API Local continuam válidas, mas o processamento passa a ser remoto quando a aplicação é linkada com a *libjidoshaLightRemote.so*.

```
//==============================================================================
// FUNCTIONS
//==============================================================================
JL_API int jidoshaLight_setRemoteSyncServerIp(
    const char* ip,
    unsigned int port
);
```

6.2.2.1 Métodos

jidoshaLight_setRemoteSyncServerIp	

Protótipo da Função	`JL_API int jidoshaLight_setRemoteSyncServerIp(`
	`  const char* ip,`
	`  unsigned int port`
	`);`

Descrição	Configura globalmente o endereço IP e porta TCP utilizada para conexão com um servidor de reconhecimento de placas. A sessão é estabelecida e fechada a cada chamada de reconhecimento.
Parâmetros	*const char* ip*: string contendo o endereço IP do servidor. *int port*: inteiro contendo a porta TCP do servidor.
Retorno	Código de retorno *JIDOSHA_LIGHT_SUCCESS* no caso de sucesso, outro código caso contrário (ver Códigos de retorno de função)

6.2.3 API JidoshaLight C/C++ (Remota Assíncrona)

A API Remota Assíncrona estende a API Local, permitindo configurar um servidor remoto que processará as imagens remotamente ao invés de as processarem localmente. Deve ser utilizada em conjunto com a biblioteca *libjidoshaLightRemote.so*.

```
//==============================================================================
//  TYPES
//==============================================================================
typedef struct JidoshaLightHandle JidoshaLightHandle;

/* Recognition result callback function pointer */
typedef void (*JCallback) (
    JidoshaLightRecognition rec,
    int rc,
    uint8_t* buffer,
    unsigned int bufferSize,
    void* arg
);

typedef struct JidoshaLightClientConfig
{
    int             queueSize;
    const char*     ip;
    int             port;
    JCallback       callback;
    void*           arg;
} JidoshaLightClientConfig;
```

```
typedef struct JidoshaLightServerInfo
{
    JidoshaLightLicenseInfo license;
    int major;
    int minor;
    int release;
} JidoshaLightServerInfo;

//==============================================================================
//  FUNCTION CALLS
//==============================================================================
//  HANDLE
//==============================================================================
JL_API JidoshaLightHandle* jl_async_create_handle(JidoshaLightClientConfig* clientConfig);
JL_API int jl_async_destroy_handle(JidoshaLightHandle* handle);
JL_API int jl_async_connect(JidoshaLightHandle* handle);
JL_API int jl_async_connect_info(JidoshaLightHandle* handle, JidoshaLightServerInfo* info);
JL_API int jl_async_get_localqueue_size(JidoshaLightHandle* handle);

//==============================================================================
//  PROCESSING
//==============================================================================
JL_API int jl_async_ANPR_fromFile (
    JidoshaLightHandle* handle,
    const char* filename,
    JidoshaLightConfig* config
);

JL_API int jl_async_ANPR_fromMemory (
    JidoshaLightHandle* handle,
    const unsigned char* buffer,
    unsigned int bufferSize,
    JidoshaLightConfig* config
);

JL_API int jl_async_ANPR_fromLuma (
    JidoshaLightHandle* handle,
    unsigned char* luma,
    int width,
    int height,
    JidoshaLightConfig* config
);

JL_API int jl_async_ANPR_fromRawImgFmt (
    JidoshaLightHandle* handle,
    const unsigned char* buffer,
    int width,
    int height,
    int stride,
    JidoshaLightRawImgFmt fmt,
    JidoshaLightConfig* config
);

JL_API int jl_async_ANPR_fromImage (
    JidoshaLightHandle* handle,
    JidoshaLightImage* img,
    JidoshaLightConfig* config
```

```
);

JL_API int jl_async_ANPR_multi_fromImage (
    JidoshaLightHandle* handle,
    JidoshaLightImage* img,
    JidoshaLightConfig* config,
    int maxPlates
);
```

6.2.3.1 Tipos

struct JidoshaLight*Handle*	
Descrição	A finalidade dessa estrutura é armazenar o objeto cliente de um servidor de reconhecimento de placas.
Membros	Nenhum

typedef void JCallback	
Descrição	A finalidade desse tipo é definir o formato da callback de usuário para o recebimento de eventos do servidor.
Membros	*struct JidoshaLightRecognition rec*: struct onde será armazenado o resultado do reconhecimento *int rc*: código de retorno da requisição (ver Códigos de retorno de função) *uint8_t* buffer*: ponteiro para a imagem onde o reconhecimento foi realizado (este ponteiro só é válido durante a execução da callback) *unsigned int bufferSize*: tamanho da imagem *void* arg*: ponteiro para estrutura opaca fornecida pelo usuário na criação do *handle*

struct JidoshaLightClientConfig	
Descrição	A finalidade dessa estrutura é definir os parâmetros da conexão entre o cliente e o servidor.
Membros	*int queueSize*: tamanho máximo de requisições pendentes para este *handle*. *const char* ip*: string contendo o endereço IP do servidor. *int port*: inteiro contendo a porta TCP do servidor. *JCallback callback*: função designada para o processamento dos resultados gerados pelo servidor. *void* arg*: ponteiro opaco para estrutura de dados de usuário utilizada para tratamento de eventos do servidor. Este ponteiro é repassado como parâmetro da callback de usuário.

struct JidoshaLightServerInfo	
Descrição	Struct utilizada para armazenar informações de licença e versão de um servidor JidoshaLight.
Membros	*JidoshaLightLicenseInfo license*: estrutura contendo informações sobre a licença do servidor - ver struct JidoshaLightLicenseInfo *int major*: valor do major da versão da biblioteca utilizada pelo servidor *int minor*: valor do minor da versão da biblioteca utilizada pelo servidor

	int release: valor do release da versão da biblioteca utilizada pelo servidor

6.2.3.2 Métodos

jl_async_create_handle	

Protótipo da Função	`JidoshaLightHandle* jl_async_create_handle(`
	`  JidoshaLightClientConfig* config`
	`);`

Descrição	Cria o *handle* de um cliente assícrono para conexão com um servidor de reconhecimento de placas.
Parâmetros	*JidoshaLightClientConfig* config*: configuração para este *handle*.
Retorno	Retorna um ponteiro para o *handle* do tipo *JidoshaLightHandle* ou *NULL* em caso de erro.

jl_async_destroy_handle	

Protótipo da Função	`int jl_async_destroy_handle(`
	`  JidoshaLightHandle* handle`
	`);`

Descrição	Desaloca o *handle* do um cliente assícrono, fechando a conexão com o servidor de reconhecimento de placas.
Parâmetros	*JidoshaLightHandle* handle*: Ponteiro para o *handle* criado por jl_async_create_handle.
Retorno	Código de retorno *JIDOSHA_LIGHT_SUCCESS* no caso de sucesso, outro código caso contrário (ver Códigos de retorno de função)

jl_async_connect	

Protótipo da Função	`int jl_async_connect(`
	`  JidoshaLightHandle* handle`
	`);`

Descrição	Estabelece a sessão com o servidor de reconhecimento de placas para um dado *handle*. Esta função bloqueia até que a conexão seja estabelecida ou ocorra timeout.
Parâmetros	*JidoshaLightHandle* handle*: Ponteiro para o *handle* criado por jl_async_create_handle.
Retorno	Código de retorno *JIDOSHA_LIGHT_SUCCESS* no caso de sucesso, outro código caso contrário (ver Códigos de retorno de função)

jl_async_connect_info	

Protótipo da Função	`int jl_async_connect_info(`
	`  JidoshaLightHandle* handle,`
	`  JidoshaLightServerInfo* info`
	`);`

Descrição	Possui a mesma funcionalidade da função jl_async_connect mas recebe um parâmetro adicional para receber informações sobre a licença e a versão do servidor.
Parâmetros	*JidoshaLightHandle* handle*: Ponteiro para o *handle* criado por jl_async_create_handle.

	JidoshaLightServerInfo info*: Ponteiro para uma struct JidoshaLightServerInfo
Retorno	Código de retorno *JIDOSHA_LIGHT_SUCCESS* no caso de sucesso, outro código caso contrário (ver Códigos de retorno de função)

jl_async_get_localqueue_size	

Protótipo da Função	`int jl_async_get_localqueue_size(`
	`  JidoshaLightHandle* handle`
	`);`

Descrição	Retorna o tamanho da fila de requisições pendentes no lado cliente para um dado *handle*.
Parâmetros	*JidoshaLightHandle* handle*: Ponteiro para o *handle* criado por jl_async_create_handle
Retorno	Retorna o número de requisições pendentes na fila local (cliente).

jl_async_ANPR_fromFile	

Protótipo da Função	`int jl_async_ANPR_fromFile(`
	`  JidoshaLightHandle* handle,`
	`  const char* filename,`
	`  JidoshaLightConfig* config`
	`);`

Descrição	Ver descrição do método jidoshaLight_ANPR_fromFile
Parâmetros	*JidoshaLightHandle* handle*: Ponteiro para o *handle* criado por jl_async_create_handle *const char* filename*: Ver descrição do método jidoshaLight_ANPR_fromFile *JidoshaLightConfig* config*: Ver descrição do método jidoshaLight_ANPR_fromFile
Retorno	Ver descrição do método jidoshaLight_ANPR_fromFile

jl_async_ANPR_fromMemory	

Protótipo da Função	`int jl_async_ANPR_fromMemory(`
	`  JidoshaLightHandle* handle,`
	`  const unsigned char* buffer,`
	`  unsigned int bufferSize,`
	`  JidoshaLightConfig* config`
	`);`

Descrição	Ver descrição do método jidoshaLight_ANPR_fromMemory
Parâmetros	*JidoshaLightHandle* handle*: Ponteiro para o *handle* criado por jl_async_create_handle *const unsigned char* buffer*: Ver descrição do método jidoshaLight_ANPR_fromMemory *unsigned int bufferSize*: Ver descrição do método jidoshaLight_ANPR_fromMemory *JidoshaLightConfig* config*: Ver descrição do método jidoshaLight_ANPR_fromMemory
Retorno	Ver descrição do método jidoshaLight_ANPR_fromMemory

jl_async_ANPR_fromLuma

Protótipo da Função	`int jl_async_ANPR_fromLuma(`
	`  JidoshaLightHandle* handle,`
	`  unsigned char* luma,`
	`  int width,`
	`  int height,`
	`  JidoshaLightConfig* config`
	`);`

Descrição	Ver descrição do método jidoshaLight_ANPR_fromLuma
Parâmetros	*JidoshaLightHandle* handle*: Ponteiro para o *handle* criado por jl_async_create_handle *unsigned char* luma*: Ver descrição do método jidoshaLight_ANPR_fromLuma *int width*: Ver descrição do método jidoshaLight_ANPR_fromLuma *int height*: Ver descrição do método jidoshaLight_ANPR_fromLuma *JidoshaLightConfig* config*: Ver descrição do método jidoshaLight_ANPR_fromLuma
Retorno	Ver descrição do método jidoshaLight_ANPR_fromLuma

jl_async_ANPR_fromRawImgFmt	

Protótipo da Função	`int jl_async_ANPR_fromRawImgFmt (`
	`  JidoshaLightHandle* handle,`
	`  const unsigned char* buffer,`
	`  int width,`
	`  int height,`
	`  int stride,`
	`  JidoshaLightRawImgFmt fmt,`
	`  JidoshaLightConfig* config,`
	`  JidoshaLightRecognition* rec`
	`);`

Descrição	Ver descrição do método jidoshaLight_ANPR_fromRawImgFmt
Parâmetros	*JidoshaLightHandle* handle*: Ponteiro para o *handle* criado por jl_async_create_handle *const unsigned char* buffer*: Ver descrição do método jidoshaLight_ANPR_fromRawImgFmt *int width*: Ver descrição do método jidoshaLight_ANPR_fromRawImgFmt *int height*: Ver descrição do método jidoshaLight_ANPR_fromRawImgFmt *int stride*: Ver descrição do método jidoshaLight_ANPR_fromRawImgFmt *JidoshaLightRawImgFmt fmt*: Ver descrição do método jidoshaLight_ANPR_fromRawImgFmt *JidoshaLightConfig* config*: Ver descrição do método jidoshaLight_ANPR_fromRawImgFmt *JidoshaLightRecognition* rec*: Ver descrição do método jidoshaLight_ANPR_fromRawImgFmt
Retorno	Ver descrição do método jidoshaLight_ANPR_fromRawImgFmt

jl_async_ANPR_fromImage	

Protótipo da Função	`int jl_async_ANPR_fromImage (`
	`  JidoshaLightHandle* handle,`
	`  JidoshaLightImage* img,`
	`  JidoshaLightConfig* config`
	`);`

Descrição	Ver descrição do método jidoshaLight_ANPR_fromImage.

Parâmetros	*JidoshaLightHandle* handle*: Ponteiro para o *handle* criado por jl_async_create_handle *JidoshaLightImage* img*: Ver descrição do método jidoshaLight_ANPR_fromImage *JidoshaLightConfig* config*: Ver descrição do método jidoshaLight_ANPR_fromImage
Retorno	Ver descrição do método jidoshaLight_ANPR_fromImage.

jl_async_ANPR_multi_fromImage	

Protótipo da Função	`int jl_async_ANPR_multi_fromImage (`
	`  JidoshaLightHandle* handle,`
	`  JidoshaLightImage* img,`
	`  JidoshaLightConfig* config,`
	`  int maxPlates`
	`);`

Descrição	Reconhece múltiplas placas a partir de uma JidoshaLightImage previamente carregada, causando múltiplas chamadas a função de callback. Um conjunto de reconhecimentos pertencentes a uma mesma imagem podem ser identificados pelo campo *int frameId* da struct JidoshaLightRecognition Ver descrição do método jidoshaLight_ANPR_multi_fromImage para maiores detalhes.
Parâmetros	*JidoshaLightHandle* handle*: Ponteiro para o *handle* criado por jl_async_create_handle *JidoshaLightImage* img*: Ver descrição do método jidoshaLight_ANPR_multi_fromImage. *JidoshaLightConfig* config*: Ver descrição do método jidoshaLight_ANPR_multi_fromImage. *int maxPlates*: Ver descrição do método jidoshaLight_ANPR_multi_fromImage
Retorno	Ver descrição do método jidoshaLight_ANPR_multi_fromImage.

6.2.4 API JidoshaLight C/C++ (Servidor)

A API Servidor estende a API Local, permitindo criar e configurar um servidor de leitura de placas para uso com as APIs remotas. Deve ser utilizada em conjunto com a biblioteca *libjidoshaLight.so*.

```
//==============================================================================
// TYPES
//==============================================================================
typedef struct JidoshaLightServer JidoshaLightServer;

typedef struct JidoshaLightServerConfig
{
    int port;
    int conns;
    int threads;
    int threadQueueSize;
    int queueTimeout;
} JidoshaLightServerConfig;

//==============================================================================
// FUNCTIONS
//==============================================================================
JL_API JidoshaLightServer* jidoshaLightServer_create(
    JidoshaLightServerConfig* serverConfig
);
```

```
JL_API int jidoshaLightServer_destroy(
    JidoshaLightServer* handler
);
```

6.2.4.1 Tipos

struct JidoshaLightServer	
Descrição	A finalidade dessa estrutura é armazenar o objeto servidor de reconhecimento de placas.
Membros	Nenhum

struct JidoshaLightServerConfig	
Descrição	A finalidade dessa estrutura é configurar o comportamento da biblioteca quando atuando como servidor de reconhecimento de placas.
Membros	*int port*: número da porta TCP utilizada para troca de mensagens. *int conns*: número de conexões clientes simultâneas aceitas pelo servidor. *int threads*: número de threads de processamento paralelo iniciadas pelo servidor. *int threadQueueSize*: tamanho máximo da fila de requisições para cada thread de processamento. *int queueTimeout*: tempo máximo de espera de uma requisição na fila de processamento em milissegundos (ms). O valor *0* indica que não há timeout.

6.2.4.2 Métodos

jidoshaLightServer_create	

Protótipo da Função	`JidoshaLightServer* jidoshaLightServer_create(`
	`  JidoshaLightServerConfig* serverConfig`
	`);`

Descrição	Cria uma instância de servidor de reconhecimento de placas. Utiliza a struct de configuração apontada por *JidoshaLightServerConfig* serverConfig* e retorna o ponteiro para *handler* do tipo *JidoshaLightServer*.
Parâmetros	*serverConfig*: ponteiro para estrutura *struct JidoshaLightServerConfig* com a configuração do servidor
Retorno	Retorna um ponteiro para o *handle* do tipo *JidoshaLightServer* ou *NULL* em caso de erro.

jidoshaLightServer_destroy	

Protótipo da Função	`int jidoshaLightServer_destroy(`
	`  JidoshaLightServer* handler`
	`);`

Descrição	Desaloca a instância de servidor identificada pelo seu *handler*.
Parâmetros	*handler*: Ponteiro para instância do servidor.
Retorno	Código de retorno *JIDOSHA_LIGHT_SUCCESS* no caso de sucesso, outro código caso contrário (ver Códigos de retorno de função)

6.3 API JidoshaLight Java

A API Java da biblioteca JidoshaLight possui variações entre as versões Linux e Android™ do SDK.

A versão Linux é um simples *wrapper* sobre a API C enquanto a versão Android™ possui funções especializadas de processamento que se enquadram melhor neste ambiente de desenvolvimento. Métodos específicos para uma ou outra plataforma estão especificados na descrição do método.

6.3.1 API JidoshaLight Java (Local)

```
public class JidoshaLight {

    //==========================================================================
    // CODES
    //==========================================================================
    /* enum JidoshaLightVehicleType */
    public static final int VEHICLE_TYPE_CAR                    = 1;
    public static final int VEHICLE_TYPE_MOTO                   = 2;
    public static final int VEHICLE_TYPE_BOTH                   = 3;

    /* enum JidoshaLightMode */
    public static final int MODE_DISABLE                        = 0;
    public static final int MODE_FAST                           = 1;
    public static final int MODE_NORMAL                         = 2;
    public static final int MODE_SLOW                           = 3;
    public static final int MODE_ULTRA_SLOW                     = 4;

    /* enum JidoshaLightCountryCode */
    public static final int COUNTRY_CODE_ARGENTINA              = 32;
    public static final int COUNTRY_CODE_BRAZIL                 = 76;
    public static final int COUNTRY_CODE_CHILE                  = 152;
    public static final int COUNTRY_CODE_COLOMBIA               = 170;
    public static final int COUNTRY_CODE_MEXICO                 = 484;
    public static final int COUNTRY_CODE_PARAGUAY               = 600;
    public static final int COUNTRY_CODE_PERU                   = 604;
    public static final int COUNTRY_CODE_URUGUAY                = 858;
    public static final int COUNTRY_CODE_NETHERLANDS            = 528;
    public static final int COUNTRY_CODE_FRANCE                 = 250;

    /* enum JidoshaLightReturnCode */
    /* success */
    public static final int SUCCESS                             = 0;
    /* basic errors */
    public static final int ERROR_FILE_NOT_FOUND                = 1;
    public static final int ERROR_INVALID_IMAGE                 = 2;
    public static final int ERROR_INVALID_IMAGE_TYPE            = 3;
    public static final int ERROR_INVALID_PROPERTY              = 4;
    public static final int ERROR_COUNTRY_NOT_SUPPORTED         = 5;
    public static final int ERROR_API_CALL_NOT_SUPPORTED        = 6;
    public static final int ERROR_INVALID_ROI                   = 7;
    public static final int ERROR_INVALID_HANDLE                = 8;
    public static final int ERROR_API_CALL_HAS_NO_EFFECT        = 9;
    public static final int ERROR_INVALID_IMAGE_SIZE            = 10;
    /* license errors */
    public static final int ERROR_LICENSE_INVALID               = 16;
    public static final int ERROR_LICENSE_EXPIRED               = 17;
```

```
    public static final int ERROR_LICENSE_MAX_THREADS_EXCEEDED    = 18;
    public static final int ERROR_LICENSE_UNTRUSTED_RTC           = 19;
    /* others */
    public static final int ERROR_OTHER                           = 999;

    /* enum JidoshaLightReturnCodeNetwork */
    /* network errors */
    public static final int ERROR_SERVER_CONNECT_FAILED           = 100;
    public static final int ERROR_SERVER_DISCONNECTED             = 101;
    public static final int ERROR_SERVER_QUEUE_TIMEOUT            = 102;
    public static final int ERROR_SERVER_QUEUE_FULL               = 103;
    public static final int ERROR_SOCKET_IO_ERROR                 = 104;
    public static final int ERROR_SOCKET_WRITE_FAILED             = 105;
    public static final int ERROR_SOCKET_READ_TIMEOUT             = 106;
    public static final int ERROR_SOCKET_INVALID_RESPONSE         = 107;
    public static final int ERROR_HANDLE_QUEUE_FULL               = 108;
    public static final int ERROR_SERVER_CONN_LIMIT_REACHED       = 213;
    public static final int ERROR_SERVER_VERSION_NOT_SUPPORTED    = 214;
    public static final int ERROR_SERVER_NOT_READY                = 215;

    /* Raw image pixel format */
    public static final int IMG_FMT_XRGB_8888                     = 0;
    public static final int IMG_FMT_RGB_888                       = 1;
    public static final int IMG_FMT_LUMA                          = 2;
    public static final int IMG_FMT_YUV420                        = 3;

    //==========================================================================
    // TYPES
    //==========================================================================
    public static class Config {
        public int vehicleType         = VEHICLE_TYPE_BOTH;
        public int processingMode      = MODE_ULTRA_SLOW;
        public int timeout             = 0;
        public int countryCode         = COUNTRY_CODE_BRAZIL;

        public float minProbPerChar       = 0.85f;
        public int maxLowProbabilityChars = 0;
        public byte lowProbabilityChar    = '?';
        public float avgPlateAngle        = 0.0f;
        public float avgPlateSlant        = 0.0f;
        public int   maxCharHeight        = 60;
        public int   minCharHeight        = 9;
        public int   maxCharWidth         = 40;
        public int   minCharWidth         = 1;
        public int   avgCharHeight        = 20;
        public int   avgCharWidth         = 7;

        public int[] xRoi              = new int[4];
        public int[] yRoi              = new int[4];
    }

    public static class Recognition {
        public String plate;
        public float[] probabilities;

        public int xText;
        public int yText;
```

```
        public int widthText;
        public int heightText;

        public int[] xChar;
        public int[] yChar;
        public int[] widthChar;
        public int[] heightChar;

        public int textColor;
        public int isMotorcycle;
        public int countryCode;

        /* JidoshaLightJidoshaLightRecognitionInfo */
        public double totalTime;
        public double localizationTime;
        public double segmentationTime;
        public double classificationTime;
        public double loadDecodeTime;
        public int[]  libVersion;
        public String libSHA1;
    }

    public static class LicenseInfo {
        public String serial;
        public String customer;
        public int maxThreads;
        public int maxConnections;
        public int state;
        public int ttl;
    }

    public static class Version {
        public int major;
        public int minor;
        public int release;
    }

    /* STATIC METHODS */
    /* PROCESSING [LINUX ONLY] */
    public static native int ANPR_fromFile(
        String filename,
        Config config,
        Recognition rec
    );

    public static native int ANPR_fromMemory(
        byte[] buffer,
        int bufferSize,
        Config config,
        Recognition rec
    );

    public static native int ANPR_fromLuma(
        byte[] luma,
        int width,
        int height,
        Config config,
```

```
        Recognition rec
    );

    /* PROCESSING [ANDROID ONLY] */
    public static native int ANPR_fromBitmap(
        Bitmap bitmap,
        Config config,
        Recognition rec
    );

    public static int ANPR_fromUri(
        Context context,
        Uri uri,
        Config config,
        Recognition rec
    );

    /* PROCESSING [LINUX AND ANDROID] */
    public static native int ANPR_fromImage(
        JidoshaLightImage img,
        Config config,
        Recognition rec
    );

    public static native int ANPR_multi_fromImage(
        JidoshaLightImage img,
        Config config,
        int maxPlates,
        List<Recognition> recList
    );

    //==========================================================================
    // LICENSE [ANDROID]
    //==========================================================================
    public static final int LICENSE_REQUEST_OK = 200;
    public static final int LICENSE_REQUEST_BAD_REQUEST = 400;
    public static final int LICENSE_REQUEST_NOT_FOUND = 404;
    public static final int LICENSE_REQUEST_UNAUTHORIZED = 401;
    public static final int LICENSE_REQUEST_FORBIDDEN = 403;
    public static final int LICENSE_REQUEST_PAYMENT_REQUIRED = 402;
    public static final int LICENSE_REQUEST_INTERNAL_SERVER_ERROR = 500;
    public static final int LICENSE_REQUEST_SERVICE_UNAVAILABLE = 503;
    public static final int LICENSE_REQUEST_ORIGIN_IS_UNREACHABLE = 523;

    public static native String getAndroidFingerprint(Activity androidActivity);
    public static native int getLicenseFromServer(Activity activity, String savePath,
String user, String key);
    public static native int setLicenseFromData(Activity androidActivity, byte[] data, int
dataSize);

    /* STATUS */
    public static native int getVersion(Version version);
    public static native String getBuildSHA1();
    public static native String getBuildFlags();

    //==========================================================================
    // LICENSE STATUS
```

```
    //==========================================================================
    public static native int getLicenseInfo(LicenseInfo info);

    //==========================================================================
    // SHARED LIBRARY LOADER
    //==========================================================================
    public static void loadLibrary() {
        System.loadLibrary("jidoshaLightJava");
    }

}
```

6.3.1.1 Tipos

class JidoshaLightImage	
Descrição	Possui as mesmas funcionalidades da struct JidoshaLightImage da API C.

Métodos públicos	`public JidoshaLightImage();`
	Constrói um novo objeto do tipo `*JidoshaLightImage*`. Caso a alocação do *handle* nativo falhe, lança uma `*RuntimeException*`.
	Para evitar vazamentos de memória, todo objeto `*JidoshaLightImage*` criado deve ser explicitamente destruído pelo usuário utilizando a função `*destroy()*`.
	
	`public JidoshaLightImage duplicate();`
	Duplica um objeto do tipo `*JidoshaLightImage*` previamente criado e carregado na memória. O novo objeto precisa ser destruído pelo usuário utilizando a função `*destroy()*`.
	
	`public int destroy();`
	Libera a memória alocada pelo objeto.
	
	`public int setLazyDecode(boolean enable);`
	Ver struct JidoshaLightImage da API C.
	
	`public int loadFromFile(String filename);`
	Ver struct JidoshaLightImage da API C.
	
	`public int loadFromMemory(byte[] buffer);`
	Ver struct JidoshaLightImage da API C.
	
	`public int loadFromRawImgFmt(byte[] buffer, int width, int height, int stride, int fmt);`
	Ver struct JidoshaLightImage da API C.

class JidoshaLight.Config	
Descrição	Possui as mesmas funcionalidades da struct JidoshaLightConfig da API C.

Membros	*int vehicleType*: indica o tipo de placa que o OCR deve buscar. Os possíveis valores para este campo são: • JidoshaLight.VEHICLE_TYPE_CAR: ver JIDOSHA_LIGHT_VEHICLE_TYPE_CAR. • JidoshaLight.VEHICLE_TYPE_MOTO: ver JIDOSHA_LIGHT_VEHICLE_TYPE_MOTO. • JidoshaLight.VEHICLE_TYPE_BOTH: ver JIDOSHA_LIGHT_VEHICLE_TYPE_BOTH. *int processingMode*: indica a estratégia de processamento adotada pelo algoritmo de reconhecimento. Os possíveis valores para este campo são: • JidoshaLight.MODE_DISABLE: ver JIDOSHA_LIGHT_MODE_DISABLE. • JidoshaLight.MODE_FAST: ver JIDOSHA_LIGHT_MODE_FAST. • JidoshaLight.MODE_NORMAL: ver JIDOSHA_LIGHT_MODE_NORMAL. • JidoshaLight.MODE_SLOW: ver JIDOSHA_LIGHT_MODE_SLOW. • JidoshaLight.MODE_ULTRA_SLOW: ver JIDOSHA_LIGHT_MODE_ULTRA_SLOW. *int timeout*: ver API C. *int countryCode*: ver API C. *float minProbPerChar*: ver API C. *int maxLowProbabilityChars*: ver API C. *byte lowProbabilityChar*: ver API C. *float avgPlateAngle*: ver API C. *float avgPlateSlant*: ver API C. *int maxCharHeight*: ver API C. *int minCharHeight*: ver API C. *int maxCharWidth*: ver API C. *int minCharWidth*: ver API C. *int avgCharHeight*: ver API C. *int avgCharWidth*: ver API C. *int xRoi[]* e *int yRoi[]*: coordenadas x e y dos quatro pontos da região de interesse da imagem (ROI - Region Of Interest). Ver API C para mais informações.

class JidoshaLight.Recognition	
Descrição	Concatena as funcionalidades dos tipos struct JidoshaLightRecognition e struct JidoshaLightRecognitionInfo da API C.
Membros	*String plate*: string contendo os caracteres da placa reconhecida ou vazia se a placa não foi encontrada. *float probabilities[]*: ver API C. *int frameId*: ver API C. *int xText* e *int yText*: ver API C. *int widthText*: ver API C. *int heightText*: ver API C. *int xChar[]* e *int yChar[]*: ver API C. *int widthChar[]*: ver API C. *int heightChar[]*: ver API C. *int textColor*: ver API C. *int isMotorcycle*: ver API C. *double totalTime*: ver API C.

	double localizationTime: ver API C. *double segmentationTime*: ver API C. *double classificationTime*: ver API C. *double loadDecodeTime*: ver API C. *int libVersion[]*: ver API C. *String libSHA1[]*: ver API C.

class JidoshaLight.LicenseInfo	
Descrição	Tipo usado pela função getLicenseInfo para retornar as informações sobre a licença da biblioteca.
Membros	*String serial*: serial number da licença em decimal *String customer*: 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 Códigos de retorno de função) *int ttl*: time-to-live em horas para licenças do tipo RTC. Este campo possui o valor **-1** caso a licença não seja expirável

class JidoshaLight.Version	
Descrição	Tipo usado pela função *getVersion* para retornar a versão da biblioteca.
Membros	*int major*: valor do major da versão. *int minor*: valor do minor da versão. *int release*: valor do release da versão.

6.3.1.2 Métodos

JidoshaLight.ANPR_fromImage [LINUX e ANDROID]	

Protótipo da Função	`public static native int ANPR_fromImage(`
	`  JidoshaLightImage img,`
	`  Config config,`
	`  Recognition rec`
	`);`

Descrição	Possui o mesmo comportamento da função jidoshaLight_ANPR_fromImage da API C.
Parâmetros	*img*: objeto do tipo JidoshaLightImage contendo a imagem a ser reconhecido. *config*: objeto do tipo JidoshaLight.Config contendo as configurações para a biblioteca. Passar `*null*` neste parâmetro implica no uso das configurações padrão da biblioteca. *rec*: objeto do tipo JidoshaLight.Recognition onde será armazenado o resultado da leitura.
Retorno	Código de retorno *JidoshaLight.SUCCESS* no caso de sucesso, outro código caso contrário (ver Códigos de retorno de função).

JidoshaLight.ANPR_multi_fromImage [LINUX e ANDROID]	

Protótipo da Função	`public static native int ANPR_multi_fromImage(`
	`  JidoshaLightImage img,`
	`  Config config,`
	`  int maxPlates,`
	`  List<Recognition> recList`
	`);`

Descrição	Possui o mesmo comportamento da função jidoshaLight_ANPR_multi_fromImage da API C.
Parâmetros	*img*: objeto do tipo JidoshaLightImage contendo a imagem a ser reconhecido. *config*: objeto do tipo JidoshaLight.Config contendo as configurações para a biblioteca. Passar `*null*` neste parâmetro implica no uso das configurações padrão da biblioteca. *maxPlates*: número máximo de placas a serem reconhecidas na imagem (1 a 8) *recList*: lista de objetos do tipo JidoshaLight.Recognition onde serão armazenados os resultados da leitura. Possui tamanho igual a maxPlates caso a função retorne com sucesso.
Retorno	Código de retorno *JidoshaLight.SUCCESS* no caso de sucesso, outro código caso contrário (ver Códigos de retorno de função).

JidoshaLight.ANPR_fromFile [LINUX]	

Protótipo da Função	`public static native int ANPR_fromFile(`
	`  String filename,`
	`  Config config,`
	`  Recognition rec`
	`);`

Descrição	Possui o mesmo comportamento da função jidoshaLight_ANPR_fromFile da API C.
Parâmetros	*filename*: string contendo o caminho para o arquivo de imagem a ser reconhecido. *config*: objeto do tipo JidoshaLight.Config contendo as configurações para a biblioteca. Passar `null` neste parâmetro implica no uso das configurações padrão da biblioteca. *rec*: objeto do tipo JidoshaLight.Recognition onde será armazenado o resultado da leitura.
Retorno	Código de retorno *JidoshaLight.SUCCESS* no caso de sucesso, outro código caso contrário (ver Códigos de retorno de função).

JidoshaLight.ANPR_fromMemory [LINUX]	

Protótipo da Função	`public static native int ANPR_fromMemory(`
	`  byte[] buffer,`
	`  int bufferSize,`
	`  Config config,`
	`  Recognition rec`
	`);`

Descrição	Possui o mesmo comportamento da função jidoshaLight_ANPR_fromMemory da API C.
Parâmetros	*buffer*: array de bytes que contém a imagem. *bufferSize*: tamanho do array de bytes.

	config: objeto do tipo JidoshaLight.Config contendo as configurações para a biblioteca. Um objeto `*null*` neste parâmetro implica no uso das configurações padrão da biblioteca. *rec*: objeto do tipo JidoshaLight.Recognition onde será armazenado o resultado da leitura.
Retorno	Código de retorno *JidoshaLight.SUCCESS* no caso de sucesso, outro código caso contrário (ver Códigos de retorno de função).

JidoshaLight.ANPR_fromLuma [LINUX]	

Protótipo da Função	`public static native int ANPR_fromLuma(`
	`  byte[] luma,`
	`  int width,`
	`  int height,`
	`  Config config,`
	`  Recognition rec`
	`);`

Descrição	Possui o mesmo comportamento da função jidoshaLight_ANPR_fromLuma da API C.
Parâmetros	*luma*: array de bytes que contém a imagem no formato RAW grayscale 8-bits. *width*: largura da imagem. *height*: altura da imagem. *config*: objeto do tipo JidoshaLight.Config contendo as configurações para a biblioteca. Um objeto `*null*` neste parâmetro implica no uso das configurações padrão da biblioteca. *rec*: objeto do tipo JidoshaLight.Recognition onde será armazenado o resultado da leitura.
Retorno	Código de retorno *JidoshaLight.SUCCESS* no caso de sucesso, outro código caso contrário (ver Códigos de retorno de função).

JidoshaLight.ANPR_fromBitmap [ANDROID]	

Protótipo da Função	`public static native int ANPR_fromBitmap (`
	`  Bitmap bitmap,`
	`  Config config,`
	`  Recognition rec`
	`);`

Descrição	Reconhece uma placa a partir do objeto `*bitmap*` utilizando as configurações presentes em `*config*`. O resultado do reconhecimento é retornado em `*rec*`. Caso ocorra algum erro no processo de reconhecimento, o objeto `*rec*` conterá uma string vazia como placa e um valor diferente de `*JidoshaLight.SUCCESS*` será retornado pela função. Os possíveis valores de retorno estão definidos na classe `*JidoshaLight*` e contém o prefixo `*ERROR_*`.
Parâmetros	*bitmap*: objeto do tipo `*android.graphics.Bitmap*` contendo a imagem a ser reconhecida no formato `*ARGB8888*`. *config*: objeto do tipo JidoshaLight.Config contendo as configurações para a biblioteca. Um objeto `*null*` neste parâmetro implica no uso das configurações padrão da biblioteca. *rec*: objeto do tipo JidoshaLight.Recognition onde será armazenado o resultado da leitura.
Retorno	Código de retorno *JidoshaLight.SUCCESS* no caso de sucesso, outro código caso contrário (ver Códigos de retorno de função)

JidoshaLight.ANPR_fromUri [ANDROID]	

Protótipo da Função	`public static int ANPR_fromUri(`
	`  Context context,`
	`  Uri uri,`
	`  Config config,`
	`  Recognition rec`
	`);`

Descrição	Reconhece uma placa a partir da `*Uri*` de um arquivo de imagem. Internamente este método chama a função ANPR_fromBitmap. O resultado do reconhecimento é retornado em `*rec*`. Caso ocorra algum erro no processo de reconhecimento, o objeto `*rec*` conterá uma string vazia como placa e um valor diferente de `*JidoshaLight.SUCCESS*` será retornado pela função. Os possíveis valores de retorno estão definidos na classe `*JidoshaLight*` e contém o prefixo `*ERROR_*`.
Parâmetros	*context*: objeto do tipo `*android.content.Context*` contendo o contexto da Activity. *uri*: objeto do tipo `*android.net.Uri*` contendo a `*uri*` da imagem a ser reconhecida. *config*: objeto do tipo JidoshaLight.Config contendo as configurações para a biblioteca. Um objeto `*null*` neste parâmetro implica no uso das configurações padrão da biblioteca. *rec*: objeto do tipo JidoshaLight.Recognition onde será armazenado o resultado da leitura.
Retorno	Código de retorno *JidoshaLight.SUCCESS* no caso de sucesso, outro código caso contrário (ver Códigos de retorno de função)

JidoshaLight.getAndroidFingerprint [ANDROID]	
Protótipo da Função	`static native String getAndroidFingerprint(Activity androidActivity);`
Descrição	Retorna o identificador único gerado pela instalação da biblioteca.
Parâmetros	*androidActivity*: objeto do tipo `*android.app.Activity*` contendo a referência para a Activity principal da aplicação.
Retorno	String contendo o identificador único da instalação necessário para a geração do arquivo de licença. Esta string não deve ser alterada.

JidoshaLight.getLicenseFromServer [ANDROID]	
Protótipo da Função	`static native int getLicenseFromServer(Activity activity, String savePath, String user, String key);`
Descrição	Requisita um arquivo de licença do servidor de licenças da Pumatronix.
Parâmetros	*activity*: objeto do tipo `*android.app.Activity*` contendo a referência para a Activity principal da aplicação. *savePath*: caminho onde deve ser salvo o arquivo de licença recebido em caso de sucesso. *user*: usuário a ser usado na requisição ou `*null*` caso contrário. *key*: chave a ser usada na requisição ou `*null*` caso contrário.
Retorno	• JidoshaLight.LICENSE_REQUEST_OK • JidoshaLight.LICENSE_REQUEST_BAD_REQUEST • JidoshaLight.LICENSE_REQUEST_NOT_FOUND • JidoshaLight.LICENSE_REQUEST_UNAUTHORIZED

	• JidoshaLight.LICENSE_REQUEST_FORBIDDEN • JidoshaLight.LICENSE_REQUEST_PAYMENT_REQUIRED • JidoshaLight.LICENSE_REQUEST_INTERNAL_SERVER_ERROR • JidoshaLight.LICENSE_REQUEST_SERVICE_UNAVAILABLE • JidoshaLight.LICENSE_REQUEST_ORIGIN_IS_UNREACHABLE Ver sample Android para maiores informações.

JidoshaLight.setLicenseFromData [ANDROID]	
Protótipo da Função	`static native int setLicenseFromData(Activity androidActivity, byte[] data, int dataSize);`
Descrição	Este método é utilizado para configurar o arquivo de licença da biblioteca a partir do conteúdo do *buffer* `*byte[] data*`.
Parâmetros	*androidActivity*: objeto do tipo `android.app.Activity` contendo a referência para a Activity principal da aplicação. *data*: buffer com o conteúdo do arquivo de licença. *dataSize*: tamanho do arquivo de licença - `*data.len*`
Retorno	Código de retorno *JidoshaLight.SUCCESS* no caso de sucesso, outro código caso contrário (ver Códigos de retorno de função)

JidoshaLight.getVersion	
Protótipo da Função	`public static native int getVersion(Version version);`
Descrição	Retorna a versão da biblioteca no formato major.minor.release.
Parâmetros	*version*: objeto do tipo JidoshaLight.Version com o número da versão
Retorno	Sempre retorna *JidoshaLight.SUCCESS*.

JidoshaLight.getBuildSHA1	
Protótipo da Função	`String getBuildSHA1();`
Descrição	Possui o mesmo comportamento da função jidoshaLight_getBuildSHA1 da API C.
Parâmetros	Nenhum
Retorno	Retorna uma String contendo o valor do SHA1 do build.

JidoshaLight.getBuildFlags	
Protótipo da Função	`String getBuildFlags();`
Descrição	Possui o mesmo comportamento da função JidoshaLight_getBuildFlags da API C.

Parâmetros	Nenhum
Retorno	Retorna uma String contendo as flags do build da biblioteca.

JidoshaLight.getLicenseInfo	
Protótipo da Função	`public static native int getLicenseInfo(LicenseInfo info);`
Descrição	Função utilizada para ler as informações da licença utilizada pela biblioteca JidoshaLight.
Parâmetros	*info*: objeto do tipo JidoshaLight.LicenseInfo
Retorno	Retorna *JIDOSHA_LIGHT_SUCCESS* em caso de sucesso.

6.3.1.3 Códigos de retorno de função

Os códigos retornados pelas funções da biblioteca JidoshaLight estão definidos como atributos `*public static final int*` dentro da classe *JidoshaLight*. Os códigos retornados nas versões Linux e ANDROID do SDK são:

- JidoshaLight.ERROR_FILE_NOT_FOUND: retornado pelas funções `*ANPR_fromFile*` e `ANPR_fromUri` quando o caminho do arquivo especificado não existe
- JidoshaLight.ERROR_INVALID_IMAGE: retornado pelas funções `*ANPR*`. Ocorre quando a imagem passada está corrompida
- JidoshaLight.ERROR_INVALID_IMAGE_TYPE: retornado pelas funções `*ANPR*`. Ocorre quando se tenta processar uma imagem de formato não suportado. Este código de erro não é retornado pela versão Android da API
- JidoshaLight.ERROR_INVALID_PROPERTY: retornado por todas as funções que possuem argumentos. Ocorre quando o argumento é inválido
- JidoshaLight.ERROR_COUNTRY_NOT_SUPPORTED: retornado pelas funções `*ANPR*` quando o código do país fornecido na estrutura de configuração não é suportado pela biblioteca
- JidoshaLight.ERROR_API_CALL_NOT_SUPPORTED: retornado quando uma função da API não está disponível para uma determinada plataforma
- JidoshaLight.ERROR_INVALID_ROI: retornado quando uma região de interesse inválida é fornecida. Ver a descrição da `struct JidoshaLightConfig` para maiores informações
- JidoshaLight.ERROR_INVALID_*HANDLE*: retornado quando o *handle* passado para a função não foi inicializado corretamente.
- JidoshaLight.ERROR_API_CALL_HAS_NO_EFFECT: retornado quando uma função da API não teve efeito ao ser executada. Pode ocorrer quando existe precedência entre chamadas.
- JidoshaLight.ERROR_LICENSE_INVALID: retornado pelas funções `*ANPR*` quando a licença fornecida não é válida (para licenças do tipo *hardkey*, significa que este não está conectado ou apresenta problemas). Contate a Pumatronix Equipamentos Eletrônicos para maiores informações
- JidoshaLight.ERROR_LICENSE_EXPIRED: retornado pelas funções `*ANPR*` quando o período de uso da licença expirou. Este tipo de erro só acontece para licenças do tipo demonstração. Contate a Pumatronix Equipamentos Eletrônicos para maiores informações
- JidoshaLight.ERROR_LICENSE_MAX_THREADS_EXCEEDED: retornado pelas funções `*ANPR*` quando o número máximo de threads concorrentes ultrapassa o permitido pela licença
- JidoshaLight.ERROR_LICENSE_UNTRUSTED_RTC: retornado pelas funções `*ANPR*` quando uma licença com data limite de uso não tem disponível uma referência confiável de tempo/data
- JidoshaLight.ERROR_OTHER: retornado quando um erro inesperado ocorre. Contate a Pumatronix Equipamentos Eletrônicos para suporte

6.3.2 API JidoshaLight Java (Remota Assíncrona)

Nota: Todas as funções da API Remota Assíncrona estão disponíveis para Android e Linux.

```
package br.gaussian.jidoshalight;

public class JidoshaLightRemote {

    //==========================================================================
    // TYPES
    //==========================================================================
    public static class Config {
        public int     queueSize;
        public String  ip;
        public int     port;
    }

    public static class ServerInfo {
        public JidoshaLight.LicenseInfo license;
        public JidoshaLight.Version version;
    }

    //==========================================================================
    // Callback interface
    //==========================================================================
    public interface CallBacks {
        void on_lpr_result_cb(JidoshaLight.Recognition rec, int code, byte[] buffer);
    }

    //==========================================================================
    // FUNCTION CALLS
    //==========================================================================
    public static native long create_handle(Config config, CallBacks callbacks);
    public static native int destroy_handle(long handle);
    public static native int connect(long handle);
    public static native int  connect_info(long handle, ServerInfo info);
    public static native int get_localqueue_size(long handle);
    public static native int ANPR_fromMemory (
        long handle,
        byte[] buffer,
        JidoshaLight.Config config
    );
    public static native int ANPR_fromRawImgFmt (
        long handle,
        byte[] buffer,
        int width,
        int height,
        int stride,
        int fmt,
        JidoshaLight.Config config
    );

    //==========================================================================
    // LIBRARY STATUS
    //==========================================================================
```

```
    public static class Version {
        public int major;
        public int minor;
        public int release;
    }

    public static native int getVersion(Version version);
    public static native String getBuildSHA1();
    public static native String getBuildFlags();

}
```

6.3.2.1 Tipos

class JidoshaLightRemote.Config	
Descrição	A finalidade dessa estrutura é armazenar o objeto cliente de um servidor de reconhecimento de placas.
Membros	*queueSize*: tamanho máximo de requisições pendentes para o *handle*. *ip*: string contendo o endereço IP do servidor. *port*: inteiro contendo a porta TCP do servidor.

interface JidoshaLightRemote.CallBacks	
Descrição	Interface que define o formato da callback para o recebimento de eventos do servidor.
Membros	*on_lpr_result_cb(JidoshaLight.Recognition rec, int code, byte[] buffer)* • *rec*: objeto contendo o resultado do reconhecimento • *code*: código de retorno da requisição • *buffer*: buffer com a imagem utilizada no reconhecimento

class JidoshaLightRemote.ServerInfo	
Descrição	Struct utilizada para armazenar informações de licença e versão de um servidor JidoshaLight.
Membros	*JidoshaLight.LicenseInfo license*: informações sobre a licença utilizada pelo servidor *JidoshaLight.Version version*: versão da biblioteca do servidor

6.3.2.2 Métodos

JidoshaLightRemote.create_*handle*	
Protótipo da Função	`long create_handle (Config config, CallBacks callbacks);`
Descrição	Cria o *handle* de um cliente assícrono para conexão com um servidor de reconhecimento de placas. Uma chamada à *destroy_handle* deve ser feita para liberar os recursos alocados.
Parâmetros	Um objeto `*JidoshaLightRemote.Config*` contendo os parâmetros de configuração do servidor e um objeto que implemente a interface `*JidoshaLightRemote.CallBacks*`.
Retorno	Em caso de sucesso, retorna o endereço de memória do *handle* criado. Caso contrário, retorna `*0*`.

JidoshaLightRemote.destroy_*handle*	
Protótipo da Função	`int destroy_handle (long handle);`
Descrição	Libera os recursos alocados para o *handle*. Para evitar que um *handle* já liberado seja reutilizado indevidamente, recomenda-se que após a chamada desta função, atribua-se `0` ao valor do *handle*, ou seja `mHandle = 0;`.
Parâmetros	Um `*long*` contendo o endereço de memória de um *handle* `*JidoshaLightRemote*` válido.
Retorno	*JidoshaLight.SUCCESS* em caso de sucesso.

JidoshaLightRemote.connect	
Protótipo da Função	`int connect (long handle);`
Descrição	Estabelece a sessão com o servidor de reconhecimento de placas para um dado *handle*.
Parâmetros	*long handle*: long contendo o endereço de memória de um *handle* JidoshaLightRemote válido.
Retorno	*JidoshaLight.SUCCESS* em caso de sucesso, caso contrário, ver códigos de errors.

JidoshaLightRemote.connect_info	
Protótipo da Função	`int connect_info(long handle, ServerInfo info);`
Descrição	Possui a mesma funcionalidade da função connect mas recebe um parâmetro adicional para receber informações sobre a licença e a versão do servidor.
Parâmetros	*long handle*: long contendo o endereço de memória de um *handle* JidoshaLightRemote válido. *ServerInfo* info*: Objeto do tipo JidoshaLightRemote.ServerInfo
Retorno	*JidoshaLight.SUCCESS* em caso de sucesso, caso contrário, ver códigos de errors.

JidoshaLightRemote.get_localqueue_size	
Protótipo da Função	`int get_localqueue_size (long handle);`
Descrição	Retorna o tamanho da fila de requisições pendentes no lado cliente para um dado *handle*.
Parâmetros	Um `*long*` contendo o endereço de memória de um *handle* JidoshaLightRemote válido.
Retorno	Retorna o número de requisições pendentes na fila local (cliente).

JidoshaLightRemote.ANPR_fromMemory	

Protótipo da Função	`int ANPR_fromMemory (`
	`  long handle,`
	`  byte[] buffer,`
	`  JidoshaLight.Config config`
	`);`

Descrição	Versão remota da chamada `JidoshaLight.ANPR_fromMemory`.
Parâmetros	*handle*: long contendo o endereço de memória de um *handle* JidoshaLightRemote válido. *buffer*: array de bytes contendo a imagem a ser reconhecida no formato JPEG, PNG ou BMP. *config*: objeto do tipo JidoshaLight.Config contendo as configurações para a biblioteca. Um objeto `null` neste parâmetro implica no uso das configurações padrão da biblioteca.
Retorno	*JidoshaLight.SUCCESS* em caso de sucesso, caso contrário, ver códigos de errors.

JidoshaLightRemote.ANPR_fromRawImgFmt	
Protótipo da Função	`int ANPR_fromRawImgFmt ( long handle, byte[] buffer, int width, int height, int stride, int fmt, JidoshaLight.Config config );`
Descrição	Envia uma requisição de reconhecimento de placa a partir de uma imagem no formato RAW.
Parâmetros	*handle*: long contendo o endereço de memória de um *handle* JidoshaLightRemote válido. *buffer*: array de bytes contendo a imagem a ser reconhecida em algum dos formatos raw suportados (ver definições na classe `JidoshaLight`). *width*: largura da imagem *height*: altura da imagem *stride*: número de bytes por linha da imagem *fmt*: formato da imagem (ver definições na classe `JidoshaLight`). *config*: objeto do tipo JidoshaLight.Config contendo as configurações para a biblioteca. Um objeto `null` neste parâmetro implica no uso das configurações padrão da biblioteca.
Retorno	*JidoshaLight.SUCCESS* em caso de sucesso, caso contrário, ver códigos de errors.

JidoshaLightRemote.getVersion	
Protótipo da Função	`public static native int getVersion(Version version);`
Descrição	Retorna a versão da biblioteca no formato major.minor.release.
Parâmetros	*version*: objeto do tipo `*Version*` com o número da versão
Retorno	Sempre retorna `JidoshaLight.SUCCESS`.

JidoshaLightRemote.getBuildSHA1	
Protótipo da Função	`String getBuildSHA1();`

Descrição	Possui o mesmo comportamento da função jidoshaLight_getBuildSHA1 da API C.
Parâmetros	Nenhum
Retorno	Retorna uma String contendo o valor do SHA1 do build.

JidoshaLightRemote.getBuildFlags	
Protótipo da Função	`String getBuildFlags();`
Descrição	Possui o mesmo comportamento da função jidoshaLight_getBuildFlags da API C.
Parâmetros	Nenhum
Retorno	Retorna uma String contendo as flags do build da biblioteca.

JidoshaLightRemote.getBuildFlags	
Protótipo da Função	`String getBuildFlags();`
Descrição	Possui o mesmo comportamento da função jidoshaLight_getBuildFlags da API C.
Parâmetros	Nenhum
Retorno	Retorna uma String contendo as flags do build da biblioteca.

6.3.3 API JidoshaLight Java (Servidor)

Nota: Todas as funções da API Servidor estão disponíveis para Android e Linux.

```
package br.gaussian.jidoshalight;

public class JidoshaLightServer {

    //===========================================================================
    // TYPES
    //===========================================================================
    public static class Config {
        public int port             = 51000;
        public int conns            = 1;
        public int threads          = 8;
        public int threadQueueSize  = 1000;
        public int queueTimeout     = 0;
    }

    //===========================================================================
    // FUNCTION CALLS
    //===========================================================================
```

```
    public static native long create_handle(Config config);
    public static native int  destroy_handle(long handle);

    //==========================================================================
    // LIBRARY STATUS
    //==========================================================================
    public static class Version {
        public int major;
        public int minor;
        public int release;
    }

    public static native int getVersion(Version version);
    public static native String getBuildSHA1();
    public static native String getBuildFlags();

}
```

6.3.3.1 Tipos

class JidoshaLightServer.Config	
Descrição	Estrutura de configuração para um servidor de leitura de placas.
Membros	*port*: porta de conexão do servidor. *conns*: número máximo de conexões simultâneas que o servidor pode aceitar (valor máximo limitado pela licença) *threads*: número máximo de threads de processamento que o servidor pode utilizar (valor máximo limtado pela licença). As threads são compartilhadas entre as conexões. *threadQueueSize*: tamanho máximo da fila de processamento de cada thread. *queueTimeout*: tempo máximo que uma requisição pode esperar na fila de processamento.

6.3.3.2 Métodos

JidoshaLightServer.create_*handle*	
Protótipo da Função	`long create_handle (Config config);`
Descrição	Cria um *handle* para o servidor de reconhecimento de placas e o inicializa. Uma chamada a `*destroy_handle*` deve ser feita para liberar os recursos alocados.
Parâmetros	Um objeto `*JidoshaLightServer.Config*` contendo os parâmetros de configuração do servidor.
Retorno	Em caso de sucesso, retorna o endereço de memória do *handle* criado. Caso contrário, retorna `*0*`.

JidoshaLightServer.destroy_*handle*	
Protótipo da Função	`void destroy_handle (long handle);`
Descrição	Libera os recursos alocados para o *handle* e interrompe o servidor. Para evitar que um *handle* já liberado seja reutilizado indevidamente, recomenda-se que após a chamada desta função, atribua-se `*0*` ao valor do *handle*, ou seja `mHandle = 0;`.

Parâmetros	Um `*long*` contendo o endereço de memória de um *handle* JidoshaLightServer válido.
Retorno	`*0*` caso o *handle* não possa ser criado. Caso contrário, retorna diferente de zero.

JidoshaLightServer.getVersion	
Protótipo da Função	`public static native int getVersion(Version version);`
Descrição	Retorna a versão da biblioteca no formato major.minor.release.
Parâmetros	*version*: objeto do tipo `*Version*` com o número da versão
Retorno	Sempre retorna `JidoshaLight.SUCCESS`.

JidoshaLightServer.getBuildSHA1	
Protótipo da Função	`String getBuildSHA1();`
Descrição	Possui o mesmo comportamento da função jidoshaLight_getBuildSHA1 da API C.
Parâmetros	Nenhum
Retorno	Retorna uma String contendo o valor do SHA1 do build.

JidoshaLightServer.getBuildFlags	
Protótipo da Função	`String getBuildFlags();`
Descrição	Possui o mesmo comportamento da função jidoshaLight_getBuildFlags da API C.
Parâmetros	Nenhum
Retorno	Retorna uma String contendo as flags do build da biblioteca.

6.3.4 API JidoshaLight Java (IO/Mjpeg)

Esta API provê um receptor de vídeo em formato MJPEG (Motion JPEG). Este formato de vídeo é amplamente utilizado por câmeras IP.

Nota: Todas as funções da API IO/Mjpeg estão disponíveis para Android e Linux.

```
package br.gaussian.io;

public class Mjpeg {

    //==========================================================================
    // Error Codes
    //==========================================================================
    public static final int JL_FRAME_QUEUE_FULL                    = 211;
```

```
    public static final int JL_LAST_FRAME_UNAVAILABLE               = 212;
    public static final int JL_MJPEG_HTTP_HEADER_OVERFLOW           = 1001;
    public static final int JL_MJPEG_HTTP_RESPONSE_NOT_OK           = 1002;
    public static final int JL_MJPEG_HTTP_CONTENT_TYPE_ERROR        = 1003;
    public static final int JL_MJPEG_HTTP_CONTENT_LENGTH_ERROR      = 1004;
    public static final int JL_MJPEG_HTTP_FRAME_BOUNDARY_NOT_FOUND  = 1005;
    public static final int JL_MJPEG_CONNECTION_CLOSED              = 1006;
    public static final int JL_MJPEG_CONNECT_FAILED                 = 1007;

    //==========================================================================
    // Config interface
    //==========================================================================
    public static class Config {
        public String url;
        public int timeout;
        public int bufferSize;
    }

    //==========================================================================
    // Callback interface
    //==========================================================================
    public interface Callbacks {
        void frame_cb(byte[] frame);
        void error_cb(int code);
    }

    public static native long   create_handle(Callbacks callbacks, Config config);
    public static native void   destroy_handle(long handle);
    public static native int    connect(long handle);
    public static native byte[] get_frame(long handle);

}
```

6.3.4.1 Tipos

class Mjpeg.Config	
Descrição	Estrutura de configuração para um fluxo Mjpeg.
Membros	*url*: String contendo a URL do fluxo Mjpeg no formato `http://<IP>[:PORT]/[PATH]`. *timeout*: máximo intervalo entre frames em milissegundos. Atrasos maiores que `timeout` são considerados como perda de conexão. (Valores recomendados: 1000 a 5000). *bufferSize*: máximo número de frames que podem ser enfileirados. Esse parâmetro deve ser maior que **0** e preferencialmente igual a **1**. Valores maiores que **1** devem ser considerados para os casos onde a chamada da callback `*frame_cb*` pode levar mais tempo que o framerate do fluxo.

interface Mjpeg.Callbacks	
Descrição	Interface que define as callbacks geradas pelo fluxo Mjpeg. **Nota 1: a execução da callback não pode tomar muito tempo (vide parâmetro `*bufferSize*`).**

	Nota 2: sob hipótese nenhuma pode-se chamar `*destroy_handle(long handle)*` dentro de uma callback.
Membros	*void frame_cb(byte[] frame)*: callback chamada sempre que um novo frame está disponível. O frame vem no formato JPEG. *void error_cb(int code)*: callback chamada sempre que ocorrer algum erro no fluxo Mjpeg (ver definição dos erros abaixo). Sempre que houver um erro de desconexão, o fluxo tentará reestabelecer a conectividade automaticamente. Para interromper o processo, basta que o usuário destrua o *handle*.

6.3.4.2 Métodos

Mjpeg.create_*handle*	

Protótipo da Função	`long create_handle (`
	`  Callbacks callbacks,`
	`  Config config`
	`);`

Descrição	Cria um *handle* para o uso nas funções da classe Mjpeg. Uma chamada a `*destroy_handle*` deve ser feita para liberar os recursos alocados.
Parâmetros	Um objeto que implementa a `*interface Mjpeg.Callbacks*` e um objeto `*Mjpeg.Config*` contendo os parâmetros de configuração do fluxo.
Retorno	Em caso de sucesso, retorna o endereço de memória do *handle* criado. Caso contrário, retorna `*0*`.

Mjpeg.destroy_*handle*	
Protótipo da Função	`void destroy_handle (long handle);`
Descrição	Libera os recursos alocados para o *handle*. Para evitar que um *handle* já liberado seja reutilizado indevidamente, recomenda-se que após a chamada desta função atribua-se `0` ao valor do *handle*, ou seja `mHandle = 0;`.
Parâmetros	Um `*long*` contendo o endereço de memória de um *handle* Mjpeg válido.
Retorno	`*0*` caso o *handle* não possa ser criado. Caso contrário, retorna diferente de zero.

Mjpeg.connect	
Protótipo da Função	`void connect (long handle);`
Descrição	Tenta estabelecer uma conexão com a URL definida na criação do *handle* Mjpeg.
Parâmetros	Um `*long*` contendo o endereço de memória de um *handle* Mjpeg válido.
Retorno	`*JidoshaLight.SUCCESS*` em caso de sucesso. `*Mjpeg.JL_MJPEG_CONNECT_FAILED*` caso a conexão não possa ser estabelecida imediatamente. Neste caso, o *handle* não tentará reconectar automaticamente, ficando a cargo do usuário chamar `*connect*` novamente em momento oportuno.

Mjpeg.get_frame	
Protótipo da Função	`byte[] get_frame(long handle)`
Descrição	Retorna o frame mais recente da fila de recepção do Mjpeg. Chamadas consecutivas a esta função podem retornar o mesmo frame caso nenhum quadro novo tenha sido recebido no intervalo.
Parâmetros	Um `*long*` contendo o endereço de memória de um *handle* Mjpeg válido.
Retorno	Um array de bytes contendo o último frame recebido em formato JPEG. Caso nenhum frame tenha sido recebido até o momento da chamada, a função retorna um array de tamanho 0 `*byteArray.length == 0*` e a callback `*error_cb*` é chamada com código `*JL_LAST_FRAME_UNAVAILABLE*`.

6.4 Guia de Migração - API 1 C/C++ JIDOSHA

O processo de migração de uma aplicação PC que utiliza a API 1 da biblioteca JIDOSHA para uma aplicação embarcada com a biblioteca JidoshaLight é simples e rápido:

1) a função `*lePlaca*` deve ser substituída pela função `*jidoshaLight_ANPR_fromFile*`;
2) a `*struct JidoshaConfig*` deve ser substituída pela `*struct JidoshaLightConfig*`;
3) a `*struct Reconhecimento*` pela `*struct JidoshaLightRecognition*`.

O usuário deve atentar aos novos campos de configuração do JidoshaLight, que devem ser necessariamente preenchidos com os valores corretos.

O exemplo a seguir mostra como obter o mesmo comportamento do JIDOSHA com o JidoshaLight:

- **JIDOSHA:**

```
#include <stdio.h>
#include "jidoshaCore.h"

int main(int argc, char* argv[])
{
    Reconhecimento rec;
    JidoshaConfig config;
    config.tipoPlaca = JIDOSHA_TIPO_PLACA_AMBOS;
    config.timeout = 1000;
    lePlaca(argv[1], &config, &rec);
    printf("placa: %s\n", rec.placa);
    return 0;
}
```

- **JidoshaLight:**

```
#include <stdio.h>
#include "anpr/api/jidosha_light_api.h"

int main(int argc, char* argv[])
{
    JidoshaLightRecognition rec;
    JidoshaLightConfig config     = {0};
    config.vehicleType            = JIDOSHA_LIGHT_VEHICLE_TYPE_BOTH;
    config.processingMode         = JIDOSHA_LIGHT_MODE_ULTRA_SLOW;
```

```
    config.timeout                = 1000;
    config.countryCode            = JIDOSHA_LIGHT_COUNTRY_CODE_BRAZIL;
    config.maxLowProbabilityChars = 0;
    config.minProbPerChar         = 0.85;
    config.lowProbabilityChar     = '?';
    jidoshaLight_ANPR_fromFile(argv[1], &config, &rec);
    printf("placa: %s\n", rec.plate);
    return 0;
}
```

Observações:

- A `*struct JidoshaLightConfig config*` é inicializada com zero `*{0}*`, garantindo que os campos `*int xRoi[4]*` e `*int yRoi[4]*` sejam zero e desabilitando o uso da ROI.
- O modo de processamento `*JIDOSHA_LIGHT_MODE_ULTRA_SLOW*` é o que mais se assemelha à estratégia de processamento utilizada pela biblioteca JIDOSHA.

O SDK acompanha um exemplo de aplicação mais detalhado.

7 APIs de usuário do JIDOSHA

Para uma maior facilidade na utilização e migração para a biblioteca JidoshaLight são disponibilizadas também as APIs do JIDOSHA através das bibliotecas `libjidoshaCore.so` e `jidoshaCore.dll`. É possível a troca da biblioteca do JIDOSHA por esses novos arquivos mantendo o mesmo comportamento (com algumas ressalvas; ver Builds especiais da API legada. Os arquivos com a interface do JIDOSHA se encontram dentro da pasta jidoshapc do SDK Windows ou Linux.

A API (Application Programming Interface) nativa do JIDOSHA está escrita em linguagem C, o que permite seu uso a partir de qualquer linguagem. O SDK também inclui bibliotecas wrapper para simplificar o uso da biblioteca a partir de .NET (C# e VB.NET), Java e Delphi. Esses wrappers simplesmente encapsulam as chamadas às funções da biblioteca, fazendo qualquer conversão necessária de parâmetros e resultados.

Toda a API C está disponível através de um único arquivo header, jidoshaCore.h, cujo conteúdo é apresentado a seguir. Uma descrição mais detalhada também é apresentada.

A biblioteca pode ser usada de duas formas: através da API 1 ou da API 2:

A API 1, que foi a primeira API do JIDOSHA, tem como principal motivação a facilidade de uso. É possível ler placas através de uma única chamda de função (`*lePlaca*` ou `*lePlacaFromMemory*`, no caso de linguagem C).

Já a API 2 foi criada para proporcionar maior flexibilidade na configuração da biblioteca e na carga de imagens. Por exemplo, é possível configurar o número mínimo de caracteres que devem ser lidos com confiabilidade boa para a placa ser considerada válida. É possível adicionar novos parâmetros de configuração à API 2 sem afetar usuários existentes da biblioteca (ou seja, estes usuários podem atualizar a DLL/.so do JIDOSHA para uma versão mais recente, sem precisar recompilar). Além disso, a API 2 permite o uso de imagens do tipo RAW, tanto grayscale como RGB/BGR. A compatibilidade com outros formatos pode ser adicionada conforme a necessidade.

Recomendamos a API 1 para quem precisa integrar o JIDOSHA à sua aplicação o mais rapidamente possível, e a API 2 para quem gostaria de maior controle sobre o funcionamento da biblioteca.

7.1 jidoshaCore.h

```
#define JIDOSHA_TIPO_PLACA_CARRO 1 /* reconhece apenas placas de nao-moto (outros veiculos)
*/
#define JIDOSHA_TIPO_PLACA_MOTO  2 /* reconhece apenas placas de moto */
#define JIDOSHA_TIPO_PLACA_AMBOS 3 /* reconhece qualquer placa */

enum jidoshaError {
    JIDOSHA_SUCCESS = 0,
    JIDOSHA_ERROR_HARDKEY_NOT_FOUND,
    JIDOSHA_ERROR_HARDKEY_NOT_AUTHORIZED,
    JIDOSHA_ERROR_FILE_NOT_FOUND,
    JIDOSHA_ERROR_INVALID_IMAGE,
    JIDOSHA_ERROR_INVALID_IMAGE_TYPE,
    JIDOSHA_ERROR_INVALID_PROPERTY,
    JIDOSHA_ERROR_COUNTRY_NOT_SUPPORTED,
    JIDOSHA_ERROR_OTHER = 999,
};

/* Parametros do OCR */
typedef struct JidoshaConfig
{
    int tipoPlaca;  /* indica o tipo de placa que o OCR deve buscar
                       use JIDOSHA_TIPO_PLACA_CARRO,
                           JIDOSHA_TIPO_PLACA_MOTO,
                       ou  JIDOSHA_TIPO_PLACA_AMBOS */
    int timeout;    /* timeout em milisegundos */
} JidoshaConfig;

/* Resultado do OCR */
typedef struct Reconhecimento
{
    char placa[8]; /* placa de 7 caracteres terminada com 0, ou string vazia se placa nao
foi encontrada */
    double probabilities[7]; /* valores de 0.0 a 1.0 indicando confiabilidade do
reconhecimento de cada caracter */
    int xText;      /* xText e yText sao o ponto da esquerda superior */
    int yText;      /*   do retangulo da placa */
    int widthText;  /* largura do retangulo da placa */
    int heightText; /* altura do retangulo da placa */
    int textColor;  /* cor do texto, 0 - escuro, 1 - claro */
    int isMotorcycle; /* 0 – nao-moto, 1 - moto */

} Reconhecimento;

/* API 1 ******************************************************************/

/* Roda o OCR a partir de um buffer contendo uma imagem codificada (JPG, BMP etc)
   retorna placa vazia caso o hardkey nao tenha sido encontrado ou eh invalido */
int lePlacaFromMemory(const unsigned char* stream, int n, JidoshaConfig* config,
Reconhecimento* rec);

/* Roda o OCR a partir de um arquivo cujo nome eh fornecido
   retorna placa vazia caso o hardkey nao tenha sido encontrado ou eh invalido */
int lePlaca(const char* filename, JidoshaConfig* config, Reconhecimento* rec);

/* Versao da bilioteca */
int getVersion(int* major, int* minor, int* release);
```

```
/* Numero serial do hardkey */
int getHardkeySerial(unsigned long* serial);

/* Estado do hardkey
   state == 0   -> nao autorizado
   state == 1   -> autorizado
   retorno == 0 -> hardkey encontrado
   retorno == 1 -> hardkey nao encontrado */
int getHardkeyState(int* state);

/* Tempo restante do hardkey de demonstracao
   days==-1 e hours==-1: hardkey nao eh demonstracao (duracao infinita)
 */
int getHardkeyRemainingTime(int* days, int* hours);


/* API 2 ******************************************************************/

/* Configuracao default da API:
    int tipoPlaca              = 3 (JIDOSHA_TIPO_PLACA_AMBOS)
    int timeout                = 0
    int minNumChars            = 7
    int maxNumChars            = 7
    int minCharWidth           = 1
    int avgCharWidth           = 7
    int maxCharWidth           = 40
    int minCharHeight          = 9
    int avgCharHeight          = 20
    int maxCharHeight          = 60
    double minPlateAngle       = -30.0
    double avgPlateAngle       = 0.0
    double maxPlateAngle       = 30.0
    double avgPlateSlant       = 0.0
    int adjustPerspective      = 0
    int autoSlope              = 1
    int autoSlant              = 1
    double minProbPerCharacter = 0.8
    char lowProbabilityChar    = '*'
    double excellentProb       = 0.95
    int ocrModel               = 1
    int checkSyntax            = 1
*/

/* Lista encadeada de reconhecimentos */
typedef struct ResultList
{
    struct ResultList* next;
    struct Reconhecimento* reconhecimento;
} ResultList;

/* Libera memoria de uma lista de reconhecimentos */
void jidoshaFreeResultList(ResultList* list);

typedef void JidoshaHandle; /* handle usado na API2 */
typedef void JidoshaImage;  /* handle para imagem alocada na API2 */
```

```
/* Inicializa handle da API2
   em processamento mulithread, deve-se usar um handle por thread */
JIDOSHACORE_API JidoshaHandle* jidoshaInit();

/* Finaliza um handle previamente alocado */
JIDOSHACORE_API int jidoshaDestroy(JidoshaHandle* handle);

/* Escreve uma propriedade de configuracao de tipo inteiro */
JIDOSHACORE_API int jidoshaSetIntProperty(JidoshaHandle* handle, const char* name, int
value);
/* Le uma propriedade de configuracao de tipo inteiro */
JIDOSHACORE_API int jidoshaGetIntProperty(JidoshaHandle* handle, const char* name, int*
value);

/* Escreve uma propriedade de configuracao de tipo double */
JIDOSHACORE_API int jidoshaSetDoubleProperty(JidoshaHandle* handle, const char* name,
double value);
/* Le uma propriedade de configuracao de tipo double */
JIDOSHACORE_API int jidoshaGetDoubleProperty(JidoshaHandle* handle, const char* name,
double* value);

/* Escreve uma propriedade de configuracao de tipo char */
JIDOSHACORE_API int jidoshaSetCharProperty(JidoshaHandle* handle, const char* name, char
value);
/* Le uma propriedade de configuracao de tipo char */
JIDOSHACORE_API int jidoshaGetCharProperty(JidoshaHandle* handle, const char* name, char*
value);

/* Roda o OCR em uma imagem carregada */
JIDOSHACORE_API int jidoshaFindFirst(JidoshaHandle* handle, JidoshaImage* image,
ResultList* list);

/* Roda o OCR em uma imagem carregada para ler da segunda placa em diante.
   A primeira placa deve ser lida por jidoshaFindFirst. */
JIDOSHACORE_API int jidoshaFindNext(JidoshaHandle* handle, JidoshaImage* image, ResultList*
list);

/* Carrega uma imagem jpg ou bmp a partir de um arquivo */
JIDOSHACORE_API int jidoshaLoadImage(const char* filename, JidoshaImage** img);

/* Carrega uma imagem jpg, bmp ou RAW (grayscale ou RGB/BGR)
   a partir de um buffer na memoria */
JIDOSHACORE_API int jidoshaLoadImageFromMemory(const unsigned char* buf, int n, int type,
int width, int height, JidoshaImage** img);

/* Libera a memoria de uma imagem carregada */
JIDOSHACORE_API int jidoshaFreeImage(JidoshaImage** img);

/* String para identificar o build da biblioteca */
JIDOSHACORE_API const char* jidoshaBuildInfo();

/* Numero de threads autorizadas */
JIDOSHACORE_API int jidoshaNumThreads();
```

7.2 API1 JIDOSHA C/C++

7.2.1 Tipos

struct JidoshaConfig	
Descrição	A finalidade dessa estrutura é configurar o comportamento da biblioteca na chamada de reconhecimento de placa.
Membros	*int tipoPlaca*: indica o tipo de placa que o OCR deve buscar, devendo ser um dentre os seguintes valores: • JIDOSHA_TIPO_PLACA_CARRO: apenas placas de carro serão procuradas, onde "carro" significa "não-moto", ou seja, inclui carros, caminhões, ônibus etc. • JIDOSHA_TIPO_PLACA_MOTO: apenas placas de moto serão procuradas. • JIDOSHA_TIPO_PLACA_AMBOS: ambas placas de moto e não-moto serão procuradas. *int timeout*: indica o tempo máximo que o reconhecimento de placa deve levar, em milisegundos. Um valor de zero indica que não há timeout. Um valor diferente de zero ajuda a manter baixo o tempo médio de processamento. O valor deve ser determinado com base na resolução da imagem e CPU utilizada.

struct Reconhecimento	
Descrição	A finalidade dessa estrutura é guardar o resultado do reconhecimento de placa, incluindo: os caracteres da placa, a confiabilidade de cada caracter, e as coordenadas da placa na imagem.
Membros	*char placa[8]*: placa de 7 caracteres terminada com 0, ou string vazia se a placa não foi encontrada. *double probabilities[7]*: valores de 0.0 a 1.0 indicando a confiabilidade, na forma de probabilidade, do reconhecimento de cada caracter. *int xText* e *int yText*: coordenadas do ponto da esquerda superior da placa, caso tenha sido encontrada. *int widthText*: largura do retângulo da placa. *int heightText*: altura do retângulo da placa. *int textColor*: cor do texto da placa, 0 - escuro, 1 - claro. *int isMotorcycle*: indica se placa é de moto, 0 - não-moto, 1 - moto.

7.2.2 Métodos

lePlaca	
Protótipo da Função	`int lePlaca(const char* filename, JidoshaConfig* config, Reconhecimento* rec);`
Descrição	Reconhece a placa e guarda-a num objeto `*Reconhecimento*`. A imagem deverá ser passada como parâmetro no formato de path de onde está localizada a imagem. Caso não seja encontrada nenhuma placa, ou caso o hardkey não esteja autorizado ou não foi encontrado, o objeto `*Reconhecimento*` conterá uma string vazia como placa. O arquivo de imagem deverá ser um bitmap, jpeg ou png.

Parâmetros	*filename*: path para o arquivo da imagem. *config*: ponteiro para a struct `*JidoshaConfig*` com a configuração para a biblioteca. *rec*: ponteiro para a struct `*Reconhecimento*` onde será armazenado o resultado da leitura.
Retorno	Código de erro: 0 (zero) no caso de sucesso, número diferente de zero caso contrário.

lePlacaFromMemory	
Protótipo da Função	`int lePlacaFromMemory(const unsigned char* stream, int n, JidoshaConfig* config, Reconhecimento*rec);`
Descrição	Reconhece a placa e guarda-a num objeto `*Reconhecimento*`. A imagem deverá ser passada como parâmetro no formato de array de bytes, e o número de bytes indicado pelo parâmetro `*n*`. Caso não seja encontrada nenhuma placa, ou caso o hardkey não esteja autorizado ou não foi encontrado, o objeto `*Reconhecimento*` conterá uma string vazia como placa. O arquivo de imagem deverá ser um bitmap, jpeg ou png.
Parâmetros	*stream*: array de bytes que contém a imagem. *n*: tamanho do array de bytes. *config*: ponteiro para a struct `JidoshaConfig` com a configuração para a biblioteca. *rec*: ponteiro para a struct `Reconhecimento` onde será armazenado o resultado da leitura.
Retorno	Código de erro: 0 (zero) no caso de sucesso, número diferente de zero caso contrário.

getVersion	
Protótipo da Função	`int getVersion(int* major, int* minor, int* release);`
Descrição	Usada para verificar a versão da biblioteca, no formato major.minor.release.
Parâmetros	*major*: ponteiro para variável int onde o major será escrito. *minor*: ponteiro para variável int onde o minor será escrito. *release*: ponteiro para variável int onde o release será escrito.
Retorno	Sempre retorna 0 (zero).

getHardkeySerial	
Protótipo da Função	`int getHardkeySerial(unsigned long* serial);`
Descrição	Usada para verificar o número serial do hardkey.
Parâmetros	*serial*: ponteiro para variável unsigned long onde o número de série do hardkey será escrito.
Retorno	Retorna 0 em caso de sucesso, 1 caso o hardkey não tenha sido encontrado.

getHardkeyState	
Protótipo da Função	`int getHardkeyState(int* state);`

Descrição	Usada para verificar o estado do hardkey. Se state é igual a 0, o hardkey não está autorizado; se state é igual a 1, o hardkey está autorizado.
Parâmetros	*state*: ponteiro para variável int o estado do hardkey será escrito.
Retorno	Retorna 0 em caso de sucesso, 1 caso o hardkey não tenha sido encontrado.

getHardkeyRemainingTime	
Protótipo da Função	`int getHardkeyRemainingTime(int* days, int* hours);`
Descrição	Usada para verificar o tempo restante para licenças de demonstração. Se days e hours são iguais a -1 não há limite de tempo.
Parâmetros	*days*: ponteiro para variável int onde será escrito o número de dias restantes. *hours*: ponteiro para variável int onde será escrito o número de horas restantes.
Retorno	Retorna 0 em caso de sucesso, 1 caso o hardkey não tenha sido encontrado.

7.3 API2 JIDOSHA C/C++

7.3.1 Tipos

struct ResultList	
Descrição	A finalidade dessa estrutura é armazenar a lista encadeada com os resultados dos processamentos das funções `*jidoshaFindFirst*` e `*jidoshaFindNext*`.
Membros	*struct ResultList* next*: ponteiro para o próximo nó na lista. NULL se o nó atual é o último. *struct Reconhecimento* reconhecimento*: ponteiro para o struct que contém um resultado de reconhecimento de placa.

typedef void JidoshaHandle	
Descrição	Tipo utilizado para representar a memória alocada para a configuração.

typedef void JidoshaImage	
Descrição	Tipo utilizado para representar a memória alocada para uma imagem.

7.3.2 Métodos

jidoshaFreeResultList	
Protótipo da Função	`void jidoshaFreeResultList(ResultList* list);`
Descrição	Libera a memória alocada para lista encadeada de resultados.
Parâmetros	*list*: ponteiro para um struct `*ResultList*`.

Retorno	Não possui retorno.

jidoshaInit	
Protótipo da Função	`JidoshaHandle* jidoshaInit();`
Descrição	Aloca memória para a configuração da biblioteca. No caso de uso multithread, cada thread deverá chamar `*jidoshaInit*` e usar seu próprio `*JidoshaHandle*`.
Parâmetros	Não possui.
Retorno	Retorna um ponteiro para um `*JidoshaHandle*` que será utilizado nas chamadas das funções subsequentes.

jidoshaDestroy	
Protótipo da Função	`int jidoshaDestroy(JidoshaHandle* handle);`
Descrição	Libera a memória alocada pela função jidoshaInit.
Parâmetros	*handle*: ponteiro para uma variável `*JidoshaHandle*`.
Retorno	JIDOSHA_SUCCESS.

jidoshaSetIntProperty	
Protótipo da Função	`int jidoshaSetIntProperty(JidoshaHandle* handle, const char* name, int value);`
Descrição	Altera o valor de uma variável do tipo int da configuração.
Parâmetros	*handle*: ponteiro para um `Jidosha*Handle*`. *name*: string que contém o nome da propriedade a ser alterada. *value*: valor que deverá ser atribuído à propriedade.
Retorno	JIDOSHA_SUCCESS caso o valor da variável seja alterado, JIDOSHA_ERROR_INVALID_PROPERTY caso a propriedade não exista ou não seja do tipo int.

jidoshaGetIntProperty	
Protótipo da Função	`int jidoshaGetIntProperty(JidoshaHandle* handle, const char* name, int* value);`
Descrição	Lê o valor de uma variável do tipo int da configuração.
Parâmetros	*handle*: ponteiro para um `*JidoshaHandle*`. *name*: string que contém o nome da propriedade a ser lida. *value*: ponteiro para variável int onde será escrito o valor da propriedade.
Retorno	JIDOSHA_SUCCESS caso o valor da variável seja lido, JIDOSHA_ERROR_INVALID_PROPERTY caso a propriedade não exista ou não seja do tipo int.

jidoshaSetDoubleProperty	
Protótipo da Função	`int jidoshaSetDoubleProperty(JidoshaHandle* handle, const char* name, double value);`
Descrição	Altera o valor de uma variável do tipo double da configuração.
Parâmetros	*handle*: ponteiro para um `*JidoshaHandle*`. *name*: string que contém o nome da propriedade a ser alterada. *value*: valor que deverá ser atribuído à propriedade.
Retorno	JIDOSHA_SUCCESS caso o valor da variável seja alterado, JIDOSHA_ERROR_INVALID_PROPERTY caso a propriedade não exista ou não seja do tipo double.

jidoshaGetDoubleProperty	
Protótipo da Função	`int jidoshaGetDoubleProperty(JidoshaHandle* handle, const char* name, double* value);`
Descrição	Lê o valor de uma variável do tipo double da configuração.
Parâmetros	*handle*: ponteiro para um `*JidoshaHandle*`. *name*: string que contém o nome da propriedade a ser lida. *value*: ponteiro para variável double onde será escrito o valor da propriedade.
Retorno	JIDOSHA_SUCCESS caso o valor da variável seja lido, JIDOSHA_ERROR_INVALID_PROPERTY caso a propriedade não exista ou não seja do tipo double.

jidoshaSetCharProperty	
Protótipo da Função	`int jidoshaSetCharProperty(JidoshaHandle* handle, const char* name, char value);`
Descrição	Altera o valor de uma variável do tipo char da configuração.
Parâmetros	*handle*: ponteiro para um `*JidoshaHandle*`. *name*: string que contém o nome da propriedade a ser alterada. *value*: valor que deverá ser atribuído à propriedade.
Retorno	JIDOSHA_SUCCESS caso o valor da variável seja alterado, JIDOSHA_ERROR_INVALID_PROPERTY caso a propriedade não exista ou não seja do tipo char.

jidoshaGetCharProperty	
Protótipo da Função	`int jidoshaGetCharProperty(JidoshaHandle* handle, const char* name, char* value);`
Descrição	Lê o valor de uma variável do tipo char da configuração.
Parâmetros	*handle*: ponteiro para um `*JidoshaHandle*`.

	name: string que contém o nome da propriedade a ser lida. *value*: ponteiro para variável char onde será escrito o valor da propriedade.
Retorno	JIDOSHA_SUCCESS caso o valor da variável seja lido, JIDOSHA_ERROR_INVALID_PROPERTY caso a propriedade não exista ou não seja do tipo char.

jidoshaFindFirst	
Protótipo da Função	`int jidoshaFindFirst(JidoshaHandle* handle, JidoshaImage* image, ResultList* list);`
Descrição	Reconhece a placa e guarda-a num objeto `Reconhecimento` que se encontra no primeiro nó do `*ResultList*`. A imagem deverá ser carregada utilizando as funções `*jidoshaLoadImage*` ou `*jidoshaLoadImageFromMemory*`. Caso não seja encontrada nenhuma placa, ou caso o hardkey não esteja autorizado ou não foi encontrado, o objeto `*Reconhecimento*` conterá uma string vazia como placa. Esta função deverá ser chamada apenas com um `*ResultList*` vazio.
Parâmetros	*handle*: ponteiro para um `*JidoshaHandle*` que contém a configuração da biblioteca. *image*: ponteiro para um `*JidoshaImage*` que contém a imagem a ser processada. *list*: ponteiro para um `*ResultList*` onde será armazenado o resultado do processamento.
Retorno	JIDOSHA_SUCCESS caso a imagem seja processada, caso contrário um outro valor do enum `*jidoshaError*`.

jidoshaFindNext	
Protótipo da Função	`int jidoshaFindNext(JidoshaHandle* handle, JidoshaImage* image, ResultList* list);`
Descrição	O objetivo desta função é permitir ao usuário reconhecer múltiplas placas numa mesma imagem. A função reconhece a placa e guarda-a num objeto `Reconhecimento` que se encontra no último nó do `*ResultList*`. A imagem deverá ser carregada utilizando as funções `*jidoshaLoadImage*` ou `*jidoshaLoadImageFromMemory*`. Caso não seja encontrada nenhuma placa, ou caso o hardkey não esteja autorizado ou não foi encontrado, o objeto `*Reconhecimento*` conterá uma string vazia como placa. Está função deverá ser chamada apenas com um `*ResultList*` anteriormente processado pela função `*jidoshaFindFirst*` ou `*jidoshaFindNext*`.
Parâmetros	*handle*: ponteiro para um `*JidoshaHandle*` que contém a configuração da biblioteca. *image*: ponteiro para um `*JidoshaImage*` que contém a imagem a ser processada. *list*: ponteiro para um `*ResultList*` onde será armazenado o resultado do processamento.
Retorno	JIDOSHA_SUCCESS caso a imagem seja processada, caso contrário um outro valor do enum `*jidoshaError*`.

jidoshaLoadImage	
Protótipo da Função	`int jidoshaLoadImage(const char* filename, JidoshaImage** img);`
Descrição	Carrega uma imagem de um arquivo e salva a referência como um JidoshaImage. O arquivo de imagem deverá ser um bitmap, jpeg ou png.

Parâmetros	*filename*: path para o arquivo da imagem. *img*: ponteiro-para-ponteiro para o struct `*JidoshaImage*` onde será armazenada a imagem.
Retorno	JIDOSHA_SUCCESS caso a imagem seja carregada corretamente, JIDOSHA_ERROR_FILE_NOT_FOUND caso o arquivo não seja encontrado ou não exista, JIDOSHA_ERROR_INVALID_IMAGE ou JIDOSHA_ERROR_INVALID_IMAGE_TYPE em caso de problemas na carga da imagem.

jidoshaLoadImageFromMemory	
Protótipo da Função	`int jidoshaLoadImageFromMemory(const unsigned char* buf, int n, int type, int width, int height, JidoshaImage** img);`
Descrição	Carrega uma imagem de um array de bytes e salva a referência como um `*JidoshaImage*`. A imagem deve estar em algum formato estruturado (bmp, jpg, png e etc.) ou raw (Grayscale 8bit, RGB ou BGR).
Parâmetros	*buf*: array de bytes contendo a imagem. *n*: tamanho do array em bytes. *type*: tipo da imagem: tipos estruturados=0, GRAY8=1, RGB=2, BGR=3. *width*: largura da imagem, ignorado se type==0. *height*: altura da imagem, ignorado se type==0. *img*: ponteiro-para-ponteiro para um `*JidoshaImage*` onde será armazenada a imagem.
Retorno	JIDOSHA_SUCCESS caso a imagem seja carregada corretamente, JIDOSHA_ERROR_FILE_NOT_FOUND caso o arquivo não seja encontrado ou não exista, JIDOSHA_ERROR_INVALID_IMAGE ou JIDOSHA_ERROR_INVALID_IMAGE_TYPE em caso de problemas na carga da imagem.

jidoshaFreeImage	
Protótipo da Função	`int jidoshaFreeImage(JidoshaImage** img);`
Descrição	Libera a memória alocada para armazenar uma imagem.
Parâmetros	*img*: ponteiro-para-ponteiro para um `*JidoshaImage*` que será desalocado.
Retorno	JIDOSHA_SUCCESS.

jidoshaBuildInfo	
Protótipo da Função	`const char* jidoshaBuildInfo();`
Descrição	Verifica as informações de build da biblioteca, sendo utilizada para verificar se a versão que está sendo executada é a esperada.
Parâmetros	Não possui.
Retorno	String constante que possui 12 ou 13 caracteres que representam o BuildInfo além de um terminador ('\0').

jidoshaNumThreads	
Protótipo da Função	`int jidoshaNumThreads();`
Descrição	Verifica o número de threads autorizadas no hardkey.
Parâmetros	Não possui.
Retorno	Número inteiro que representa quantas threads estão autorizadas a executarem simultaneamente as funções de OCR da biblioteca. Retorna 1 caso o hardkey não seja encontrado.

7.3.3 API 2 - Configuração

Nesta seção detalhamos todos os parâmetros de configuração disponíveis na API 2. Vale para a API C, Java, .NET e Python.

Parâmetro *tipoPlaca*	
Descrição	Serve para restringir o tipo de placa veicular que deve ser reconhecido. É interessante principalmente para reduzir o tempo de processamento. Em particular, quando `*tipoPlaca=JIDOSHA_TIPO_PLACA_CARRO*`, um método mais rápido de localização de placa pode ser utilizado pela biblioteca. Os valores válidos são:
Nome	*tipoPlaca*
Tipo	int
Valor default	JIDOSHA_TIPO_PLACA_AMBOS
Outros valores	• JIDOSHA_TIPO_PLACA_CARRO` == 1 • JIDOSHA_TIPO_PLACA_MOTO` == 2 • JIDOSHA_TIPO_PLACA_AMBOS` == 3

Parâmetro *timeout*	
Descrição	Após `*timeout*` milisegundos desde o início de processamento de uma imagem a busca da placa será encerrada e será retornada a melhor placa encontrada. Caso `*timeout*` seja zero, não há timeout. Recomenda-se utilizar um `*timeout*` diferente de zero quando a aplicação exige que as imagens sejam processadas rapidamente (com baixa latência) ou quando a carga da CPU está muito elevada.
Nome	*timeout*
Tipo	int
Valor default	0

Parâmetro *minNumChars*	
Descrição	Indica o número mínimo de caracteres que uma placa deve ter. Caso a versão em uso da biblioteca tenha múltiplas sintaxes de placa habilitadas (por exemplo, placas de múltiplos

	países), este parâmetro é ignorado, devendo-se utilizar o `*numAllowedBadChars*` no seu lugar.
Nome	*minNumChars*
Tipo	int
Valor default	7

Parâmetro *numAllowedBadChars*	
Descrição	Indica o número máximo de caracteres faltantes que uma placa pode ter, usa-se esse parâmetro quando se deseja que placas parcialmente reconhecidas sejam retornadas.
Nome	*numAllowedBadChars*
Tipo	int
Valor default	0

Parâmetro *maxNumChars*	
Descrição	Indica o número máximo de caracteres que uma placa deve ter. Atualmente este parâmetro é ignorado.
Nome	*maxNumChars*
Tipo	int
Valor default	7

Parâmetro *minCharWidth*	
Descrição	Largura mínima que um caractere deve ter, em pixels.
Nome	*minCharWidth*
Tipo	int
Valor default	1

Parâmetro *avgCharWidth*	
Descrição	Largura média esperada de um caractere, em pixels. Atualmente este parâmetro não é utilizado.
Nome	*avgCharWidth*
Tipo	int
Valor default	1

Parâmetro *maxCharWidth*	
Descrição	Largura máxima que um caractere deve ter, em pixels.
Nome	*maxCharWidth*
Tipo	int
Valor default	7

Parâmetro *minCharHeight*	
Descrição	Altura mínima que um caractere deve ter, em pixels.
Nome	*minCharHeight*
Tipo	int
Valor default	9

Parâmetro *avgCharHeight*	
Descrição	Altura média esperada de um caractere, em pixels. Esse parâmetro pode ser usado quando as placas são muito grandes. Quando `*avgCharHeight > 30*`, a imagem será reduzida internamente antes de ser processada. Os limites mínimos e máximos de tamanho do caractere serão ajustados de acordo com o fator de redimensionamento.
Nome	*avgCharHeight*
Tipo	int
Valor default	20

Parâmetro *maxCharHeight*	
Descrição	Altura máxima de um caractere, em pixels.
Nome	*maxCharHeight*
Tipo	int
Valor default	60

Parâmetro *ocrModel*	
Descrição	Define o modelo de OCR a ser utilizado no reconhecimento de caracteres. Este parâmetro existe para permitir facilmente trocar o modelo de OCR para modelos de versões anteriores da biblioteca, sem necessidade de recompilar a aplicação do usuário ou trocar a biblioteca. Não use valores diferentes do default, exceto quando recomendado pela equipe de suporte da Pumatronix Equipamentos Eletrônicos.

Nome	*ocrModel*
Tipo	int
Valor default	1

Parâmetro *checkSyntax*	
Descrição	Quando `*checkSyntax=1*` a biblioteca aplica uma etapa de processamento adicional para verificar se os caracteres reconhecidos têm a sintaxe esperada (letra ou número), o que reduz a incidência de reconhecimentos falsos (textos que não são placas). Observação: mesmo quando `*checkSyntax=0*`, a biblioteca nunca retornará um reconhecimento com sintaxe diferente da definida. Por exemplo, para placas brasileiras, a placa retornada sempre terá 3 letras seguidas de 4 números. Porém, um texto não-placa, como "*ESCOLAR*", pode ser confundido com uma placa, o que resultaria em um reconhecimento como "*ESC0148*". A sintaxe está de acordo com uma placa brasileira, apesar de não ser uma placa. Usando `*checkSyntax=1*` pode ajudar a descartar reconhecimentos falsos como no exemplo.
Nome	*checkSyntax*
Tipo	int
Valor default	1

Parâmetro *minPlateAngle*	
Descrição	Ângulo de inclinação mínimo em graus permitido para uma placa. Para mais detalhes, consulte a seção de configuração de perspectiva da imagem.
Nome	*minPlateAngle*
Tipo	double
Valor default	-30.0

Parâmetro *maxPlateAngle*	
Descrição	Ângulo de inclinação máximo em graus permitido para uma placa. Para mais detalhes, consulte a seção de configuração de perspectiva da imagem.
Nome	*maxPlateAngle*
Tipo	double
Valor default	30.0

Parâmetro *minProbPerCharacter*

Descrição	Probabilidade (confiabilidade) mínima exigida no reconhecimento de cada caractere. É extremamente importante para o bom funcionamento do OCR, e não se recomenda mudar a configuração default. No entanto, em casos específicos pode ser interessante ajustá-lo. Se `*minProbPerCharacter*` for menor que o default, o número de placas que não são reconhecidas reduzirá, mas em contrapartida o número de placas com algum caractere errado poderá aumentar. Se `*minProbPerCharacter*` for maior que o default, o número de placas que não são reconhecidas poderá aumentar, mas o número de erros será menor.
Nome	*minProbPerCharacter*
Tipo	double
Valor default	0.8

Parâmetro *excellentProb*	
Descrição	Este parâmetro existe para reduzir o tempo de processamento médio. Se todos os caracteres reconhecidos tiverem probabilidade maior ou igual a `*excellentProb*`, o reconhecimento será considerado como excelente e retornado ao usuário imediatamente, sem processamento adicional. Caso contrário, o processamento continuará até que uma das seguintes condições seja atingida: um reconhecimento excelente seja encontrado; o timeout seja atingido; ou não haja mais etapas de processamento a fazer. Valores maiores de `*excellentProb*` resultam em maiores índices de reconhecimento e menores índices de erro (confusão entre caracteres), porém com maior tempo de processamento. Valores menores de `*excellentProb*` resultam em menores índices de reconhecimento e maiores índices de erro (confusão entre caracteres), porém com menor tempo de processamento.
Nome	*excellentProb*
Tipo	double
Valor default	0.95

Parâmetro *lowProbabilityChar*	
Descrição	Caractere de substituição a ser utilizado quando um caractere da placa é reconhecido com probabilidade menor que `*minProbPerCharacter*`. Terá efeito apenas se `*minNumChars*` for menor que `*maxNumChars*`. Por exemplo, se `*lowProbabilityChar*='-'` e `*minNumChars*=6`, a placa `"ABC1234"` será retornada como `"A-C1234"` se a probabilidade do segundo caractere for menor que `*minProbPerCharacter*`.
Nome	*lowProbabilityChar*
Tipo	char
Valor default	'*'

Parâmetro *country*	
Descrição	Caractere de substituição a ser utilizado quando um caractere da placa é reconhecido com probabilidade menor que `*minProbPerCharacter*`. Terá efeito apenas se `*minNumChars*` for menor que `*maxNumChars*`. Por exemplo, se `*lowProbabilityChar='-'* e `*minNumChars=6*`, a placa `"ABC1234"` será retornada como `*"A-C1234"*` se a probabilidade do segundo caractere for menor que `*minProbPerCharacter*`.
Nome	*country*
Tipo	int
Valor default	76

7.3.4 API 2 - Configuração de perspectiva da imagem

De maneira geral, recomenda-se que a instalação da câmera para captura de placas veiculares seja feita de forma que as placas fiquem alinhadas aos eixos horizontal e vertical da imagem. No entanto, em algumas situações isso não é possível, e acaba-se obtendo placas inclinadas em relação aos eixos da imagem, o que pode prejudicar o reconhecimento das placas. Nesses casos pode-se informar à biblioteca a perspectiva da placa. A biblioteca efetuará então uma correção da perspectiva, de forma a maximizar o índice de reconhecimento de placas.

No caso de um equipamento com várias câmeras recomenda-se criar um `*handle*` da API 2 por câmera (através da função `*jidoshaInit*`) e configurar os parâmetros de perspectiva individualmente para cada `*handle*`.

Os parâmetros `*avgPlateAngle*`, `*avgPlateSlant*` e `*adjustPerspective*` são usados para informar a perspectiva da placa na imagem (inclinação horizontal e vertical) e corrigi-la. A inclinação horizontal (`*avgPlateAngle*`) e a inclinação vertical (`*avgPlateSlant*`) devem ser medidas em imagens típicas da instalação.

Além da configuração manual da perspectiva, é possível também habilitar na biblioteca algoritmos que procuram corrigir automaticamente a perspectiva. Veja os parâmetros `*autoSlope*` e `*autoSlant*` para mais detalhes.

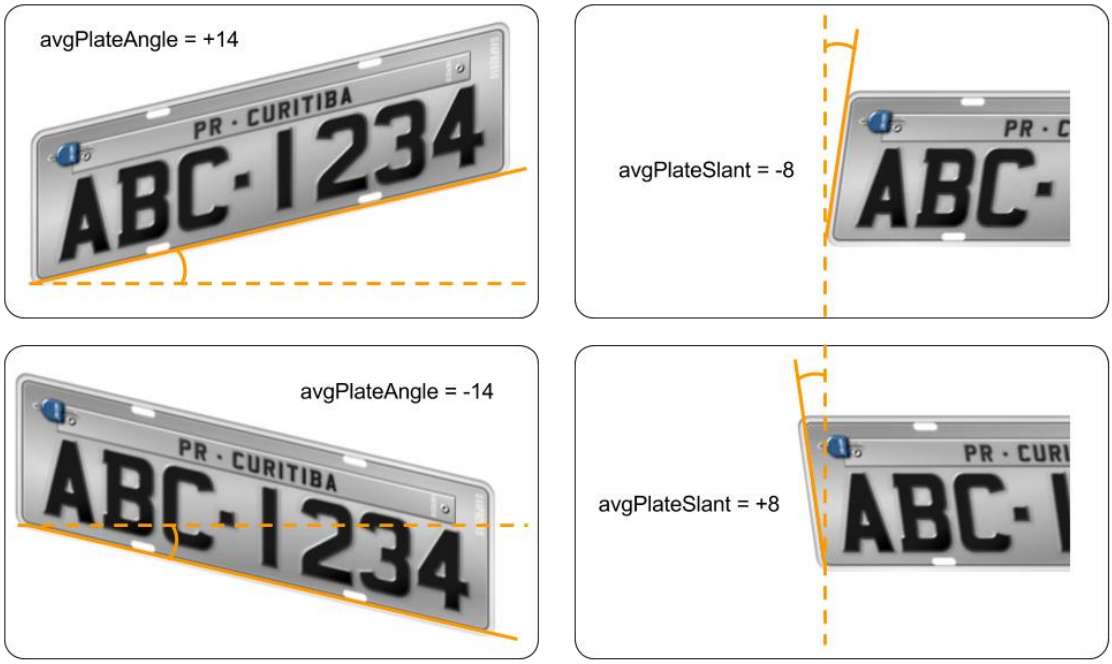


Figura 11 - Como calcular os valores de avgPlateAngle e avgPlateSlant

Parâmetro *avgPlateAngle*	
Descrição	Ângulo de inclinação horizontal médio em graus esperado para uma placa. É usado para efetuar ajuste de perspectiva da imagem. Só terá efeito se `*adjustPerspective*` for diferente de zero. O ângulo deve ser medido conforme a convenção da imagem acima.
Nome	*avgPlateAngle*
Tipo	double
Valor default	0.0

Parâmetro *avgPlateSlant*	
Descrição	Ângulo de inclinação vertical médio em graus esperado para uma placa. É usado para efetuar ajuste de perspectiva da imagem. Só terá efeito se `*adjustPerspective*` for diferente de zero. O ângulo deve ser medido conforme a convenção da imagem acima.
Nome	*avgPlateSlant*
Tipo	double
Valor default	0.0

Parâmetro *adjustPerspective*	
Descrição	`*adjustPerspective=1*` habilita o ajuste de perspectiva configurado através de `*avgPlateAngle*` e `*avgPlateSlant*`. `*adjustPerspectiva=0*` desabilita o ajuste de perspectiva (`*avgPlateAngle*` e `*avgPlateSlant*` são ignorados).

Nome	*adjustPerspective*
Tipo	int
Valor default	0

Parâmetro *autoSlope*	
Descrição	`*autoSlope=1*` habilita o ajuste automático da inclinação horizontal da placa. Caso seja usado em conjunto com o ajuste de perspectiva manual (`*avgPlateAngle*` quando `*adjustPerspective=1*`), o ajuste manual será aplicado antes do algoritmo de ajuste automático. `*autoSlope=0*` desabilita o ajuste automático da inclinação horizontal da placa.
Nome	*autoSlope*
Tipo	int
Valor default	1

Parâmetro *autoSlant*	
Descrição	`*autoSlant=1*` habilita o ajuste automático da inclinação vertical da placa. Caso seja usado em conjunto com o ajuste de perspectiva manual (`*avgPlateSlant*` quando `*adjustPerspective=1*`), o ajuste manual será aplicado antes do algoritmo de ajuste automático. `*autoSlant=0*` desabilita o ajuste automático da inclinação vertical da placa.
Nome	*autoSlant*
Tipo	int
Valor default	1

7.4 API JIDOSHA C# / VB.NET

A API .NET da biblioteca apresenta três funções overloaded, que facilitam o reconhecimento de placa a partir de três fontes: um array de bytes contendo a imagem codificada (JPG ou BMP), um objeto do tipo `*Image*`, ou um nome de arquivo. Todas necessitam como parâmetro um objeto `*JidoshaConfig*` que serve para configurar o comportamento da biblioteca.

7.4.1 API 1

7.4.1.1 Métodos

reconhecePlaca 1	
Protótipo da Função	`Reconhecimento reconhecePlaca(byte[] array, JidoshaConfig config)`
Descrição	Retorna um objeto `*Reconhecimento*` que representa o resultado de reconhecimento da placa. A imagem (JPG, BMP etc.) deve ser passada como um array de bytes.
Retorno	Objeto `*Reconhecimento*` contendo a string que representam a placa do veículo, um array de doubles contendo as probabilidades dos caracteres, as coordenadas do texto da placa, a cor do texto (escuro ou claro), e um campo indicando se a placa é de moto. Caso não seja encontrada nenhuma placa, ou caso o hardkey não esteja autorizado ou não foi encontrado, o objeto `*Reconhecimento*` conterá uma string vazia como placa.

reconhecePlaca 2	
Protótipo da Função	`Reconhecimento reconhecePlaca(Image image, JidoshaConfig config)`
Descrição	Retorna um objeto `*Reconhecimento*` que representa o resultado de reconhecimento da placa. A imagem deverá ser passada como parâmetro no formato de um objeto `*Image*`.
Retorno	Objeto `*Reconhecimento*` contendo a string que representam a placa do veículo, um array de doubles contendo as probabilidades dos caracteres, as coordenadas do texto da placa, a cor do texto (escuro ou claro), e um campo indicando se a placa é de moto. Caso não seja encontrada nenhuma placa, ou caso o hardkey não esteja autorizado ou não foi encontrado, o objeto `*Reconhecimento*` conterá uma string vazia como placa.

reconhecePlaca 3	
Protótipo da Função	`Reconhecimento reconhecePlaca(string filename, JidoshaConfig config)`
Descrição	Retorna um objeto `*Reconhecimento*` que representa o resultado de reconhecimento da placa. A imagem deverá ser passada como parâmetro no formato de path de onde está localizada a imagem.
Retorno	Objeto `*Reconhecimento*` contendo a string que representam a placa do veículo, um array de doubles contendo as probabilidades dos caracteres, as coordenadas do texto da placa, a cor do texto (escuro ou claro), e um campo indicando se a placa é de moto. Caso não seja encontrada nenhuma placa, ou caso o hardkey não esteja autorizado ou não foi encontrado, o objeto `*Reconhecimento*` conterá uma string vazia como placa.

getVersionString	
Protótipo da Função	`String getVersionString()`
Descrição	Usada para verificar a versão da biblioteca, no formato major.minor.release.
Retorno	Retorna uma string formatada com a versão.

getHardkeySerial

Protótipo da Função	`int getHardkeySerial()`
Descrição	Usada para verificar o número serial do hardkey.
Retorno	Retorna um int contendo o número serial do hardkey.

getHardkeyState	
Protótipo da Função	`int getHardkeyState()`
Descrição	Usada para verificar o estado do hardkey. Se state é igual a 0, o hardkey não está autorizado; se state é igual a 1, o hardkey está autorizado.
Retorno	Retorna o estado do hardkey (0 ou 1, conforme descrição acima).

7.4.2 Exemplos API JIDOSHA C# / VB.NET

7.4.2.1 Exemplo C#

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Drawing;

using JidoshaNET;

namespace JidoshaSample
{
    class JidoshaSample
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Jidosha build {0}", Jidosha.jidoshaBuildInfo());
            Console.WriteLine("Hardkey serial {0}", Jidosha.getHardKeySerial());
            Console.WriteLine("Hardkey {0}", Jidosha.getHardKeyState() == 1 ? "autorizado"
: "não autorizado");

            if (args.Length < 1)
            {
                Console.WriteLine("uso: jidoshaNETSample imagem");
                Console.WriteLine("Aperte Enter para sair");
                Console.ReadLine();
                return;
            }

            // Carrega a imagem
            string filename = args[0];
            Image image = Image.FromFile(filename);
            System.IO.MemoryStream stream = new System.IO.MemoryStream();
            image.Save(stream, image.RawFormat);
            byte[] array = stream.ToArray();
            stream.Close();
```

```
            stream.Dispose();

            // Sample API1
            JidoshaConfig cfg = new JidoshaConfig();
            cfg.timeout = 0;
            cfg.tipoPlaca = TipoPlaca.AMBOS;
            Reconhecimento r = Jidosha.reconhecePlaca(filename, cfg);
            System.Console.WriteLine("reconhecePlaca: {0}", r.placa);
            r = Jidosha.reconhecePlaca(array, cfg);
            System.Console.WriteLine("reconhecePlacaFromMemory: {0}", r.placa);

            // Sample API2

            // Inicializa
            IntPtr JidoshaHandle = Jidosha.jidoshaInit();

            // SetProperty
            Jidosha.jidoshaSetIntProperty(JidoshaHandle, "avgCharHeight", 20);
            Jidosha.jidoshaSetIntProperty(JidoshaHandle, "minNumChars", 6);
            Jidosha.jidoshaSetDoubleProperty(JidoshaHandle, "minProbPerCharacter", 0.7);
            Jidosha.jidoshaSetCharProperty(JidoshaHandle, "lowProbabilityChar", '_');

            // GetProperty
            int maxCharHeight = 0;
            double minProb = 0;
            maxCharHeight = Jidosha.jidoshaGetIntProperty(JidoshaHandle, "avgCharHeight");
            minProb = Jidosha.jidoshaGetDoubleProperty(JidoshaHandle,
"minProbPerCharacter");

            Console.WriteLine("Altura media: {0}", maxCharHeight);
            Console.WriteLine("Probabilidade minima: {0}", minProb);

            // Carrega uma imagem
            IntPtr JidoshaImg = Jidosha.jidoshaLoadImage(array, 0, 0, 0);

            // Reconhece placa
            ResultList resultList = new ResultList();
            Jidosha.jidoshaFindFirst(JidoshaHandle, JidoshaImg, ref resultList);
            while (resultList.reconhecimento[resultList.reconhecimento.Count - 1].placa !=
"")
            {
                Jidosha.jidoshaFindNext(JidoshaHandle, JidoshaImg, ref resultList);
            }

            // Imprime o resultado
            foreach (Reconhecimento rec in resultList.reconhecimento)
            {
                Console.WriteLine("Placa: {0}", rec.placa);
                Console.Write("Probs:");
                foreach (double d in rec.probabilities)
                    Console.Write(" {0},", d);
                Console.WriteLine("");
            }

            // Apaga a lista de reconhecimentos
            Jidosha.jidoshaFreeResultList(resultList);
```

```
            // Libera a imagem
            Jidosha.jidoshaFreeImage(JidoshaImg);

            // Libera o handle do jidosha
            Jidosha.jidoshaDestroy(JidoshaHandle);

            Console.WriteLine("Aperte Enter para sair");
            Console.ReadLine();
        }
    }
}
```

7.4.2.2 Exemplo VB.NET

```
Imports JidoshaNET

Module Module1

    Sub Main()
        Dim args() As String = Environment.GetCommandLineArgs()
        Dim filename As String = args(1)
        Dim config As JidoshaConfig = New JidoshaConfig()
        config.tipoPlaca = TipoPlaca.AMBOS
        config.timeout = 1000
        Dim rec As Reconhecimento = Jidosha.reconhecePlaca(filename, config)
        Console.WriteLine("placa: " + rec.placa)
    End Sub

End Module
```

7.5 API JIDOSHA Delphi

7.5.1 API 1

7.5.1.1 Métodos

reconhecePlaca	
Protótipo da Função	`function reconhecePlaca(filename: String; config: JidoshaConfig) : Reconhecimento;`
Descrição	Retorna um objeto `*Reconhecimento*` que representa o resultado de reconhecimento da placa. A imagem deverá ser passada como parâmetro no formato de path de onde está localizada a imagem.
Retorno	Objeto `*Reconhecimento*` contendo a string que representam a placa do veículo, um array de doubles contendo as probabilidades dos caracteres, as coordenadas do texto da placa, a cor do texto (escuro ou claro), e um campo indicando se a placa é de moto. Caso não seja encontrada nenhuma placa, ou caso o hardkey não esteja autorizado ou não foi encontrado, o objeto `*Reconhecimento*` conterá uma string vazia como placa.

reconhecePlacaFromMemory

Protótipo da Função	`function reconhecePlacaFromMemory(byteArray: array of byte; config: JidoshaConfig) : Reconhecimento;`
Descrição	Retorna um objeto `*Reconhecimento*` que representa o resultado de reconhecimento da placa. A imagem (JPG, BMP etc.) deve ser passada como um array de bytes.
Retorno	Objeto `*Reconhecimento*` contendo a string que representam a placa do veículo, um array de doubles contendo as probabilidades dos caracteres, as coordenadas do texto da placa, a cor do texto (escuro ou claro), e um campo indicando se a placa é de moto. Caso não seja encontrada nenhuma placa, ou caso o hardkey não esteja autorizado ou não foi encontrado, o objeto `*Reconhecimento*` conterá uma string vazia como placa.

7.5.2 Exemplo API JIDOSHA Delphi

Observação: Este exemplo é para Delphi 2007. Em versões mais recentes do Delphi, pode ser necessário converter a string do filepath para AnsiString antes de passar para a biblioteca C. Pode também ser necessário converter a string da placa de AnsiString para Unicode.

```
program JidoshaDelphiSample;

{$APPTYPE CONSOLE}

uses
  SysUtils,
  jidoshaDelphi in 'jidoshaDelphi.pas';

var
    filename: String;
    rec: Reconhecimento;
    config: JidoshaConfig;
begin
    if ParamCount < 1
    then begin
      Writeln('uso: jidoshaDelphiSample.exe imagem.jpg');
      Exit;
    end;

    filename := ParamStr(1);
    Writeln(filename);

    config.tipoPlaca := JIDOSHA_TIPO_PLACA_AMBOS;
    config.timeout := 1000;
    rec := reconhecePlaca(filename, config);
    Writeln('placa: ', rec.placa);
end.
```

7.6 API JIDOSHA Java

7.6.1 API 1

7.6.1.1 Métodos

reconhecePlaca	
Protótipo da Função	`public static native Reconhecimento reconhecePlaca(String filename, JidoshaConfig config);`
Descrição	Retorna um objeto `*Reconhecimento*` que representa o resultado de reconhecimento da placa. A imagem deverá ser passada como parâmetro no formato de path de onde está localizada a imagem.
Retorno	Objeto `*Reconhecimento*` contendo a string que representam a placa do veículo, um array de doubles contendo as probabilidades dos caracteres, as coordenadas do texto da placa, a cor do texto (escuro ou claro), e um campo indicando se a placa é de moto. Caso não seja encontrada nenhuma placa, ou caso o hardkey não esteja autorizado ou não foi encontrado, o objeto `*Reconhecimento*` conterá uma string vazia como placa.

reconhecePlacaFromMemory	
Protótipo da Função	`public static native Reconhecimento reconhecePlacaFromMemory(byte[] buf, JidoshaConfig config);`
Descrição	Retorna um objeto `*Reconhecimento*` que representa o resultado de reconhecimento da placa. Este objeto contém a string da placa e a probabilidade (confiabilidade) de cada caractere reconhecido. A imagem deverá ser passada como parâmetro no formato de array de bytes.
Retorno	Objeto `*Reconhecimento*` contendo a string que representam a placa do veículo, um array de doubles contendo as probabilidades dos caracteres, as coordenadas do texto da placa, a cor do texto (escuro ou claro), e um campo indicando se a placa é de moto. Caso não seja encontrada nenhuma placa, ou caso o hardkey não esteja autorizado ou não foi encontrado, o objeto `*Reconhecimento*` conterá uma string vazia como placa.

7.6.2 Exemplo API JIDOSHA Java

```
import br.com.gaussian.jidosha.Jidosha;
import br.com.gaussian.jidosha.JidoshaConfig;
import br.com.gaussian.jidosha.Reconhecimento;

class JidoshaSample {
    public static void main(String args[]) throws java.io.IOException {
        JidoshaConfig config = new JidoshaConfig(JidoshaConfig.JIDOSHA_TIPO_PLACA_AMBOS,
0);
        for (int i=0; i < args.length; i++) {
            System.out.println(args[i]);
            Reconhecimento rec = Jidosha.reconhecePlaca(args[i], config);
            System.out.println("placa: " + rec.placa);
        }
    }
}
```

7.7 Builds especiais da API legada

Por diversos motivos, a biblioteca JIDOSHA possuía diferentes tipos de builds para o mesmo número de versão, que de forma geral não são compatíveis entre si. O build pode ser verificado pelo retorno da função

`*jidoshaBuildInfo*`. A string do buildInfo tem o seguinte formato: "hash_build", onde "hash" é o hash do commit, e "build" é uma string denotando o tipo de build.

Até a v3.4.0 o JidoshaLight é compatível apenas com o build `*std*` do JIDOSHA, que é o build padrão. A partir da v3.5.0 o JidoshaLight é compatível também com o build `*charpos*` ("character positions"), desde que exista um chave no registro ou variável de ambiente, conforme detalhado abaixo. A única diferença da versão `*std*` para a versão `*charpos*` consiste em quatro campos adicionais no struct `*Reconhecimento*` no header `*jidoshaCore.h*`, que contém as coordenadas dos caracteres da placa quando o reconhecimento é bem sucedido. Essa diferença na API faz com que os builds `*std*` e `*charpos*` sejam incompatíveis (um executável compilado para um desses builds não pode ser usado com outro).

Nota: O modo de compatibilidade para o build 'charpos' é suportado apenas na API de linguagem C.

Para referência, as structs dos builds `*std*` e `*charpos*` estão listadas a seguir.

7.7.1.1 Build std

```
typedef struct Reconhecimento
{
    char placa[7+1];
    double probabilities[7];
    int xText;
    int yText;
    int widthText;
    int heightText;
    int textColor;
    int isMotorcycle;
} Reconhecimento;
```

7.7.1.2 Build charpos

```
typedef struct Reconhecimento
{
    char placa[7+1];
    double probabilities[7];
    int xText;
    int yText;
    int widthText;
    int heightText;
    int xChar[7];
    int yChar[7];
    int widthChar[7];
    int heightChar[7];
    int textColor;
    int isMotorcycle;
} Reconhecimento;
```

Para ativar o modo de compatibilidade com o build `*charpos*` no **Windows**, é necessário criar uma chave no registro do Windows, em `*HKLM|SOFTWARE|PUMATRONIX*`, com nome `*JL_LEGACY_API_TYPE*`, de tipo `*REG_SZ*`, e valor `*charpos*`. Qualquer outro valor fará com que o JidoshaLight volte ao

comportamento padrão (compatibilidade com build `*std*`). Ao invés do registro, pode-se usar uma variável de ambiente, com nome `*JL_LEGACY_API_TYPE*` e valor `*charpos*`.

A chave no registro pode ser criada com o seguinte comando no prompt (é necessário ter credenciais de Administrador):

```
REG ADD HKLM\SOFTWARE\PUMATRONIX /v JL_LEGACY_API_TYPE /t REG_SZ /d charpos /f
```

Para desligar o modo de compatibilidade com o build `*charpos*`, altere o valor da variável para uma string vazia, ou simplesmente apague a chave:

```
REG DELETE HKLM\SOFTWARE\PUMATRONIX /v JL_LEGACY_API_TYPE
```

Para ativar o modo de compatibilidade com o build `*charpos*` no **Linux**, é necessário criar uma variável de ambiente, com nome `*JL_LEGACY_API_TYPE*` e valor `*charpos*`. Qualquer outro valor fará com que o JidoshaLight volte ao comportamento padrão (compatibilidade com build `*std*`).

Observações:

- Caso o modo de compatibilidade com o build `*charpos*` esteja ativado (`*JL_LEGACY_API_TYPE=charpos*`), mas o código de usuário esteja por engano utilizando a struct `*Reconhecimento*` do build `*std*`, poderá ocorrer acesso inválido à memória ou corrupção silenciosa de dados.
- Recomenda-se migrar para a API do JidoshaLight assim que possível.

PUMATRONIX

www.pumatronix.com