Manual do Usuário - JIDOSHA

Biblioteca de Software para Reconhecimento Automático de Placas Veiculares

Versão 1.6.4

Última atualização: 2017/05

---

1 Visão Geral

1.1 Condições Gerais

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

1.2 Licença de software

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

---

2 Introdução

Este presente documento Manual do Usuário – JIDOSHA tem por objetivo detalhar as funções da biblioteca de software especializada em reconhecimento de placas veiculares, chamada JIDOSHA, e suas condições de uso para correto funcionamento.

2.1 Objetivo

A biblioteca de software JIDOSHA tem como principal funcionalidade reconhecer placas veiculares brasileiras a partir de imagens. Sua principal aplicação é a fiscalização eletrônica de trânsito, cenário para qual o JIDOSHA foi criado e no qual apresenta um excelente desempenho. Porém, é possível usar a biblioteca em qualquer tipo de controle e gestão de passagem de veículos.

Com um alto índice de reconhecimento, o JIDOSHA é a ferramenta ideal para quem necessita ter informação de placas veiculares de forma automática, sem intervenções externas, através de métodos de análise de imagem.

2.2 Condições de Uso

A biblioteca de software JIDOSHA foi criada para funcionar em conjunto com o *hardkey* (chave de segurança) que acompanha a biblioteca. Ou seja, para correto funcionamento da biblioteca o referido *hardkey* deverá estar conectado à USB do ambiente onde a biblioteca estará sendo utilizada. Há duas versões de *hardkey*, uma de demonstração e outra liberada. A versão demonstração tem data de validade, enquanto a versão liberada não. Quando a data de validade expira, a bilbioteca automaticamente passa a retornar placas vazias. Se seu *hardkey* de demonstração expirou e você deseja comprar uma licença ou estender o período de demonstração, entre em contato com a GAUSSIAN Inteligência Computacional (contato@gaussian.com.br).

Esta versão do JIDOSHA possui compatibilidade com sistema operacional Windows com plataforma Delphi, C++ Builder 6, Visual C++, .NET 2.0 ou superior, e Java; e com sistema operacional Linux com plataforma C (gcc), C++ (g++) e Java.

2.3 Instalação

2.3.0.1 Windows

Para instalar a biblioteca de software JIDOSHA, basta descompactar o arquivo de SDK (Software Development Kit) fornecido, que deverá conter DLLs e código fonte de demonstração, e copiar as DLLs para a pasta do seu projeto. Caso alguma senha seja exigida ao tentar descompactar o arquivo, a senha padrão é “jidosha”.

Para testar o funcionamento da biblioteca e do *hardkey*, plugue o *hardkey* (o Windows irá instalar um driver automaticamente na primeira vez), abra uma janela do prompt de comando (cmd.exe) e, de dentro da pasta onde descompactou o SDK, digite “jidoshaSample.exe 0 Foto_teste.jpg”. Em algumas versões da biblioteca o executável pode chamar-se “jidoshapc.exe”.

Se tudo correr bem, esse programa deve retornar, entre outras informações, a palavra “autorizado” e a placa contida na imagem. Caso a palavra “autorizado” não apareça, experimente remover o *hardkey* e plugá-lo novamente. Caso isso não resolva o problema, sua versão do *hardkey* pode ter expirado, ou pode haver algum problema com o mesmo. Nesse caso, entre em contato com a GAUSSIAN Inteligência Computacional.

2.3.0.2 Linux

Para seu correto funcionamento, as permissões do *hardkey* USB devem ser alteradas. Adicione a seguinte linha:

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

ao final do arquivo correspondente a sua distribuição Linux:

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

Já para Debian, adicione as linhas:

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

ao final do arquivo:

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

Observação: nas versões do JIDOSHA anteriores à 1.5.25, no Centos e openSUSE também é necessário adicionar o usuário que vai rodar o JIDOSHA ao grupo "lock", devido ao JIDOSHA criar um arquivo de lock em `/var/lock`. Na versão 1.5.25 ou mais recente o arquivo de lock não é mais utilizado.

Para instruções de como habilitar o *hardkey* em outras distribuições Linux, entre em contato com a GAUSSIAN Inteligência Computacional.

Após adicionar a linha ao arquivo correspondente, plugue o *hardkey* e execute `./jidoshaSample 0 Foto_teste.jpg` de dentro do diretório do SDK, o que deverá informar a versão da biblioteca, o número de série do *hardkey*, o status do *hardkey* (autorizado ou não autorizado) e, por fim, a placa da imagem de teste.

---

3 API JIDOSHA (C/C++)

A API (Application Programming Interface) nativa da biblioteca está escrita em linguagem C, o que permite seu uso a partir de qualquer linguagem. O SDK 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 comentado 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.

Início do arquivo 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              = 0
    int autoSlant              = 0
    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*

/* 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();
```

Final do arquivo jidoshaCore.h

3.1 API 1

3.1.1 struct JidoshaConfig

3.1.1.1 Descrição

A finalidade dessa estrutura é configurar o comportamento da biblioteca na chamada de reconhecimento de placa.

3.1.1.2 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.

3.1.2 struct Reconhecimento

3.1.2.1 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.

3.1.2.2 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.

3.1.3 lePlaca

3.1.3.1 Protótipo da Função

```
int lePlaca(const char* filename, JidoshaConfig* config, Reconhecimento* rec);
```

3.1.3.2 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.

3.1.3.3 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.

3.1.3.4 Retorno

Código de erro: 0 (zero) no caso de sucesso, número diferente de zero caso contrário.

3.1.4 lePlacaFromMemory

3.1.4.1 Protótipo da Função

```
int lePlacaFromMemory(const unsigned char* stream, int n, JidoshaConfig* config, Reconhecimento*rec);
```

3.1.4.2 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.

3.1.4.3 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.

3.1.4.4 Retorno

Código de erro: 0 (zero) no caso de sucesso, número diferente de zero caso contrário.

3.1.5 getVersion

3.1.5.1 Protótipo da Função

```
int getVersion(int* major, int* minor, int* release);
```

3.1.5.2 Descrição

Usada para verificar a versão da biblioteca, no formato major.minor.release.

3.1.5.3 Parâmetros

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

3.1.5.4 Retorno

Sempre retorna 0 (zero).

3.1.6 getHardkeySerial

3.1.6.1 Protótipo da Função

```
int getHardkeySerial(unsigned long* serial);
```

3.1.6.2 Descrição

Usada para verificar o número serial do hardkey.

3.1.6.3 Parâmetros

`serial`: ponteiro para variável unsigned long onde o número de série do hardkey será escrito.

3.1.6.4 Retorno

Retorna 0 em caso de sucesso, 1 caso o hardkey não tenha sido encontrado.

3.1.7 getHardkeyState

3.1.7.1 Protótipo da Função

```
int getHardkeyState(int* state);
```

3.1.7.2 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.

3.1.7.3 Parâmetros

`state`: ponteiro para variável int o estado do hardkey será escrito.

3.1.7.4 Retorno

Retorna 0 em caso de sucesso, 1 caso o hardkey não tenha sido encontrado.

3.1.8 getHardkeyRemainingTime

3.1.8.1 Protótipo da Função

```
int getHardkeyRemainingTime(int* days, int* hours);
```

3.1.8.2 Descrição

Usada para verificar o tempo restante para licenças de demonstração. Se days e hours são iguais a -1 não há limite de tempo.

3.1.8.3 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.

3.1.8.4 Retorno

Retorna 0 em caso de sucesso, 1 caso o hardkey não tenha sido encontrado.

3.2 API 2 - Tipos e Funções

3.2.1 struct ResultList

3.2.1.1 Descrição

A finalidade dessa estrutura é armazenar a lista encadeada com os resultados dos processamentos das funções `jidoshaFindFirst` e `jidoshaFindNext`.

3.2.1.2 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.

3.2.2 jidoshaFreeResultList

3.2.2.1 Protótipo da Função

```
void jidoshaFreeResultList(ResultList* list);
```

3.2.2.2 Descrição

Libera a memória alocada para lista encadeada de resultados.

3.2.2.3 Parâmetros

`list`: ponteiro para um struct `ResultList`.

3.2.2.4 Retorno

Não possui retorno.

3.2.3 typedef void JidoshaHandle

3.2.3.1 Descrição

Tipo utilizado para representar a memória alocada para a configuração.

3.2.4 typedef void JidoshaImage

3.2.4.1 Descrição

Tipo utilizado para representar a memória alocada para uma imagem.

3.2.5 jidoshaInit

3.2.5.1 Protótipo da Função

```
JidoshaHandle* jidoshaInit();
```

3.2.5.2 Descrição

Aloca memória para a configuração da biblioteca. No caso de uso multithread, cada thread deverá chamar `jidoshaInit` e usar seu próprio `JidoshaHandle`.

3.2.5.3 Retorno

Retorna um ponteiro para um `JidoshaHandle` que será utilizado nas chamadas das funções subsequentes.

3.2.6 jidoshaDestroy

3.2.6.1 Protótipo da Função

```
int jidoshaDestroy(JidoshaHandle* handle);
```

3.2.6.2 Descrição

Libera a memória alocada pela função jidoshaInit.

3.2.6.3 Parâmetros

`handle`: ponteiro para uma variável `JidoshaHandle`.

3.2.6.4 Retorno

JIDOSHA_SUCCESS.

3.2.7 jidoshaSetIntProperty

3.2.7.1 Protótipo da Função

```
int jidoshaSetIntProperty(JidoshaHandle* handle, const char* name, int value);
```

3.2.7.2 Descrição

Altera o valor de uma variável do tipo int da configuração.

3.2.7.3 Parâmetros

`handle`: ponteiro para um `JidoshaHandle`.

`name`: string que contém o nome da propriedade a ser alterada.

`value`: valor que deverá ser atribuido à propriedade.

3.2.7.4 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.

3.2.8 jidoshaGetIntProperty

3.2.8.1 Protótipo da Função

```
int jidoshaGetIntProperty(JidoshaHandle* handle, const char* name, int* value);
```

3.2.8.2 Descrição

Lê o valor de uma variável do tipo int da configuração.

3.2.8.3 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.

3.2.8.4 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.

3.2.9 jidoshaSetDoubleProperty

3.2.9.1 Protótipo da Função

```
int jidoshaSetDoubleProperty(JidoshaHandle* handle, const char* name, double value);
```

3.2.9.2 Descrição

Altera o valor de uma variável do tipo double da configuração.

3.2.9.3 Parâmetros

`handle`: ponteiro para um `JidoshaHandle`.

`name`: string que contém o nome da propriedade a ser alterada.

`value`: valor que deverá ser atribuido à propriedade.

3.2.9.4 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.

3.2.10 jidoshaGetDoubleProperty

3.2.10.1 Protótipo da Função

```
int jidoshaGetDoubleProperty(JidoshaHandle* handle, const char* name, double* value);
```

3.2.10.2 Descrição

Lê o valor de uma variável do tipo double da configuração.

3.2.10.3 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.

3.2.10.4 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.

3.2.11 jidoshaSetCharProperty

3.2.11.1 Protótipo da Função

```
int jidoshaSetCharProperty(JidoshaHandle* handle, const char* name, char value);
```

3.2.11.2 Descrição

Altera o valor de uma variável do tipo char da configuração.

3.2.11.3 Parâmetros

`handle`: ponteiro para um `JidoshaHandle`.

`name`: string que contém o nome da propriedade a ser alterada.

`value`: valor que deverá ser atribuido à propriedade.

3.2.11.4 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.

3.2.12 jidoshaGetCharProperty

3.2.12.1 Protótipo da Função

```
int jidoshaGetCharProperty(JidoshaHandle* handle, const char* name, char* value);
```

3.2.12.2 Descrição

Lê o valor de uma variável do tipo char da configuração.

3.2.12.3 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.

3.2.12.4 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.

3.2.13 jidoshaFindFirst

3.2.13.1 Protótipo da Função

```
int jidoshaFindFirst(JidoshaHandle* handle, JidoshaImage* image, ResultList* list);
```

3.2.13.2 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.

3.2.13.3 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.

3.2.13.4 Retorno

JIDOSHA_SUCCESS caso a imagem seja processada, caso contrário um outro valor do enum `jidoshaError`.

3.2.14 jidoshaFindNext

3.2.14.1 Protótipo da Função

```
int jidoshaFindNext(JidoshaHandle* handle, JidoshaImage* image, ResultList* list);
```

3.2.14.2 Descrição

O objetivo desta função é permitir ao usuário reconhecer mútliplas placas numa mesma image. 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`.

3.2.14.3 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.

3.2.14.4 Retorno

JIDOSHA_SUCCESS caso a imagem seja processada, caso contrário um outro valor do enum `jidoshaError`.

3.2.15 jidoshaLoadImage

3.2.15.1 Protótipo da Função

```
int jidoshaLoadImage(const char* filename, JidoshaImage** img);
```

3.2.15.2 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.

3.2.15.3 Parâmetros

`filename`: path para o arquivo da imagem.

`img`: ponteiro-para-ponteiro para o struct `JidoshaImage` onde será armazenada a imagem.

3.2.15.4 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.

3.2.16 jidoshaLoadImageFromMemory

3.2.16.1 Protótipo da Função

```
int jidoshaLoadImageFromMemory(const unsigned char* buf, int n, int type, int width, int height, JidoshaImage** img);
```

3.2.16.2 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).

3.2.16.3 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.

3.2.16.4 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.

3.2.17 jidoshaFreeImage

3.2.17.1 Protótipo da Função

```
int jidoshaFreeImage(JidoshaImage** img);
```

3.2.17.2 Descrição

Libera a memória alocada para armazenar uma imagem.

3.2.17.3 Parâmetros

`img`: ponteiro-para-ponteiro para um `JidoshaImage` que será desalocado.

3.2.17.4 Retorno

JIDOSHA_SUCCESS.

3.2.18 jidoshaBuildInfo

3.2.18.1 Protótipo da Função

```
const char* jidoshaBuildInfo();
```

3.2.18.2 Descrição

Verifica as informações de build da biblioteca, sendo utilizada para verificar se a versão que está sendo executada é a esperada.

3.2.18.3 Retorno

String constante que possui 12 ou 13 caracteres que representam o BuildInfo além de um terminador ('\0').

3.2.19 jidoshaNumThreads

3.2.19.1 Protótipo da Função

```
int jidoshaNumThreads();
```

3.2.19.2 Descrição

Verifica o número de threads autorizadas no hardkey.

3.2.19.3 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.

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.

3.3.1 int tipoPlaca (default: JIDOSHA_TIPO_PLACA_AMBOS)

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:

- `JIDOSHA_TIPO_PLACA_CARRO == 1`
- `JIDOSHA_TIPO_PLACA_MOTO  == 2`
- `JIDOSHA_TIPO_PLACA_AMBOS == 3 (default).`

3.3.2 int timeout (default: 0)

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.

3.3.3 int minNumChars (default: 7)

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.

3.3.4 int numAllowedBadChars (default: 0)

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.

3.3.5 int maxNumChars (default: 7)

Indica o número máximo de caracteres que uma placa deve ter. Atualmente este parâmetro é ignorado.

3.3.6 int minCharWidth (default: 1)

Largura mínima que um caractere deve ter, em pixels.

3.3.7 int avgCharWidth (default: 1)

Largura média esperada de um caractere, em pixels. Atualmente este parâmetro não é utilizado.

3.3.8 int maxCharWidth (default: 7)

Largura máxima que um caractere deve ter, em pixels.

3.3.9 int minCharHeight (default: 9)

Altura mínima que um caractere deve ter, em pixels.

3.3.10 int avgCharHeight (default: 20)

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.

3.3.11 int maxCharHeight (default: 60)

Altura máxima de um caractere, em pixels.

3.3.12 double minPlateAngle (default: -30.0)

Ângulo de inclinação mínimo em graus permitido para uma placa. Para mais detalhes, consulte a seção de configuração de perspectiva da imagem.

3.3.13 double maxPlateAngle (default: 30.0)

Ângulo de inclinação máximo em graus permitido para uma placa. Para mais detalhes, consulte a seção de configuração de perspectiva da imagem.

3.3.14 double minProbPerCharacter (default: 0.8)

Probabilidade (confiabilidade) mínima exigida no reconhecimento de cada caractere. É extremamente importante para o bom funcionamento do OCR, e não recomenda-se 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.

3.3.15 char lowProbabilityChar (default: '*')

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`.

3.3.16 double excellentProb (default: 0.95)

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.

3.3.17 int ocrModel (default: 1)

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 GAUSSIAN.

3.3.18 int checkSyntax (default: 1)

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.

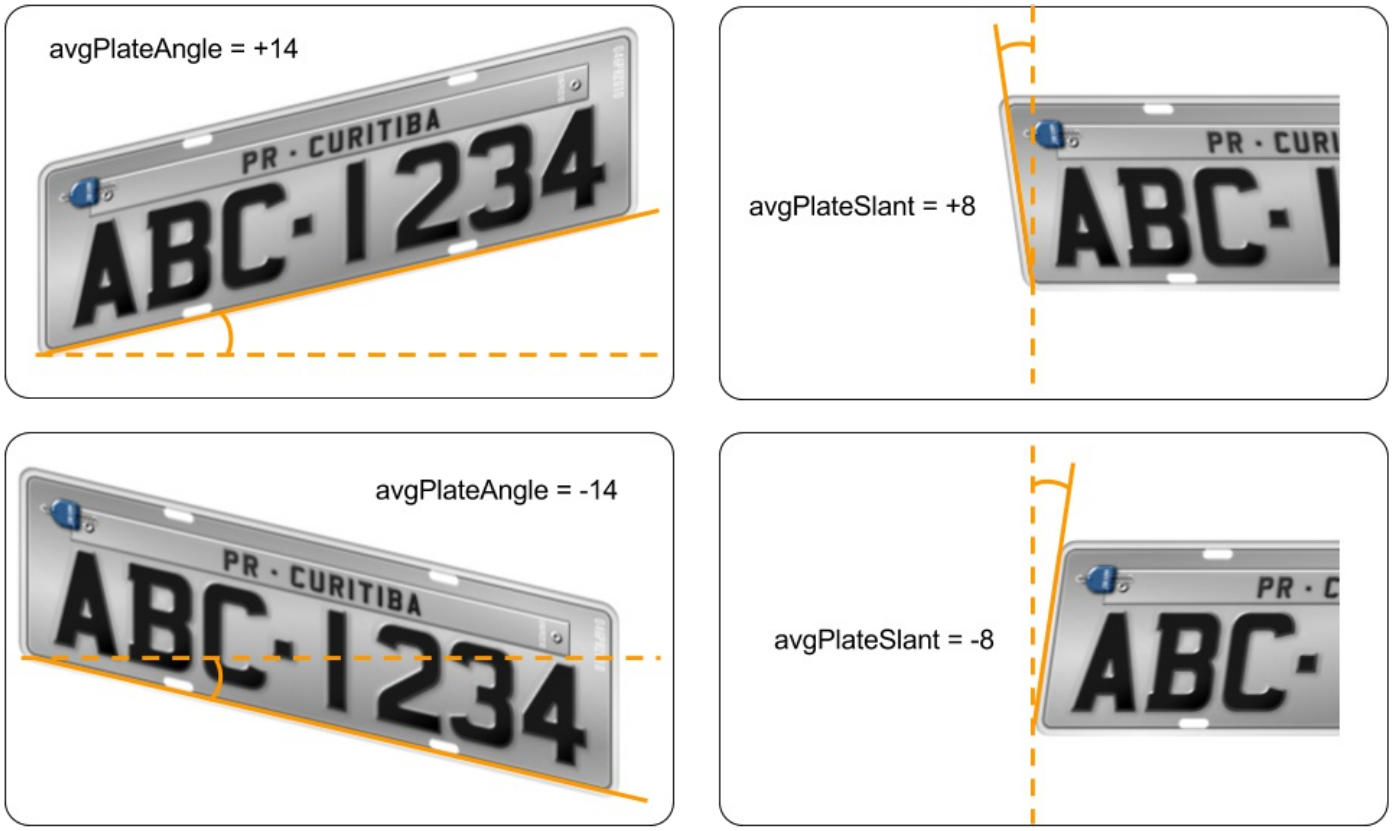
3.4 API 2 - Configuração de perspectiva da imagem

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.



3.4.1 double avgPlateAngle (default: 0.0)

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.

3.4.2 double avgPlateSlant (default: 0.0)

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.

3.4.3 int adjustPerspective (default: 0)

`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).

3.4.4 int autoSlope (default: 0)

`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.

3.4.5 int autoSlant (default: 0)

`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.

---

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.

4.1 API 1

4.1.1 reconhecePlaca 1

4.1.1.1 Protótipo da Função

```
Reconhecimento reconhecePlaca(byte[] array, JidoshaConfig config)
```

4.1.1.2 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.

4.1.1.3 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.

4.1.2 reconhecePlaca 2

4.1.2.1 Protótipo da Função

```
Reconhecimento reconhecePlaca(Image image, JidoshaConfig config)
```

4.1.2.2 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`.

4.1.2.3 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.

4.1.3 reconhecePlaca 3

4.1.3.1 Protótipo da Função

```
Reconhecimento reconhecePlaca(string filename, JidoshaConfig config)
```

4.1.3.2 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.

4.1.3.3 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.

4.1.4 getVersion

4.1.4.1 Protótipo da Função

```
String getVersionString()
```

4.1.4.2 Descrição

Usada para verificar a versão da biblioteca, no formato major.minor.release.

4.1.4.3 Retorno

Retorna uma string formata com a versão.

4.1.5 getHardkeySerial

4.1.5.1 Protótipo da Função

```
int getHardkeySerial()
```

4.1.5.2 Descrição

Usada para verificar o número serial do hardkey.

4.1.5.3 Retorno

Retorna um int contendo o número serial do hardkey.

4.1.6 getHardkeyState

4.1.6.1 Protótipo da Função

```
int getHardkeyState()
```

4.1.6.2 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.

4.1.6.3 Retorno

Retorna o estado do hardkey (0 ou 1, conforme descrição acima).

---

5 API JIDOSHA (Delphi)

5.1 API 1

5.1.1 reconhecePlaca

5.1.1.1 Protótipo da Função

```
function reconhecePlaca(filename: String; config: JidoshaConfig) : Reconhecimento;
```

5.1.1.2 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.

5.1.1.3 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.

5.1.2 reconhecePlacaFromMemory

5.1.2.1 Protótipo da Função

```
function reconhecePlacaFromMemory(byteArray: array of byte; config: JidoshaConfig) : Reconhecimento;
```

5.1.2.2 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.

5.1.2.3 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.

---

6 API JIDOSHA (Java)

6.1 API 1

6.1.1 reconhecePlaca

6.1.1.1 Protótipo da Função

```
public static native Reconhecimento reconhecePlaca(String filename, JidoshaConfig config);
```

6.1.1.2 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.

6.1.1.3 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.

6.1.2 reconhecePlacaFromMemory

6.1.2.1 Protótipo da Função

```
public static native Reconhecimento reconhecePlacaFromMemory(byte[] buf, JidoshaConfig config);
```

6.1.2.2 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.

6.1.2.3 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 Exemplos de utilização

7.1 Exemplo C/C++

```c
#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;
}
```

7.2 Exemplo C#

```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();
        }
    }
}
```

7.3 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
```

7.4 Exemplo 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.
```

7.5 Exemplo 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);
        }
    }
}
```
